NASA Conference Publication 2394

Space Station Software Recommendations

Edited by
Susan Voigt
NASA Langley Research Center
Hampton, Virginia

Report of an open forum with
industry and academia held at
NASA Marshall Space Flight Center
Huntsville, Alabama
April 24-25, 1985

NASA
National Aeronautics
and Space Administration

Scientific and Technical
Information Branch

1985

PREFACE

This forum was held to solicit industry and academic opinion on the plans that NASA is formulating to manage the development, acquisition, operation, and maintenance of software in support of the Space Station Program.

A NASA workshop, held in August 1984, identified major software issues that should be addressed early in the Space Station definition phase. These are documented in NASA CP-2361 (1985), and formed the basis for discussions at this forum.

Four major topics were selected for emphasis during the discussions and experts in these areas were invited to speak and to serve on the panels which met in closed sessions to identify important issues and in open sessions to air their views and to solicit audience comments.

As a result of the two days of discussion, several recommendations were formulated and are presented herein. These recommendations are addressed to the Space Station Program Office and are intended to help in the policy-making and management of Space Station Program software.

This publication presents the recommendations made by the forum participants and does not represent any official NASA position. These recommendations are being considered by the Space Station Program at the time of publication of this report.

CONTENTS

PRECEDING PAGE BLANK NOT FILMED

omit to
p3

FORUM ORGANIZING COMMITTEE

General chairman: Susan Voigt, NASA LaRC

Local arrangements: John Wolfsberger, NASA MSFC

Program co-chairs: Frank McGarry, NASA GSFC
James L. Raney, NASA JSC

NASA Headquarters sponsor: Dana Hall

NASA Level B sponsor: Dane Dixon

Program Committee members:

Jody Steinbacher, JPL
Robert Jackson, MITRE
John McLeod, JPL

Other participating members of Space Station Software Working Group:

Rhoda Hornstein, NASA Headquarters Code T
Robert W. Nelson, NASA GSFC
Audrey Dorofee, NASA KSC
Ed Ng, JPL
David Callender, JPL
James Soeder, NASA LeRC
Tom Purer, NASA KSC
Ai Fang, NASA Headquarters Code E
Bill Wilson, NASA Headquarters Code D

PANEL MEMBERS

Software Management Panel:

Robert Loesh, System Technology Institute, co-chair
John Manley, Computing Technology Transition, Inc., co-chair
Robert Braslau, TRW
Dana Hall, NASA Headquarters
Harlan Mills, IBM
John (Jack) Munson, System Development Corporation
Donald Reifer, Reifer Consultants, Inc.
Jody Steinbacher, Jet Propulsion Laboratory

Software Development Environment Panel:

Barry Boehm, TRW, co-chair
Tom Velez, Computer Technology Associates, co-chair
M. Dane Dixon, NASA Johnson Space Center
J. R. (Bob) Elston, Boeing Aerospace Company
Robert Loveless, Science Applications International Corp.
Frank McGarry, NASA Goddard Space Flight Center
William M. Murray, General Dynamics
James L. Raney, NASA Johnson Space Center
Richard Taylor, University of California at Irvine

Languages Panel:

Victor R. Basili, University of Maryland, co-chair
Charles McKay, University of Houston at Clear Lake, co-chair
J. Barton DeWolf, Charles Stark Draper Laboratory
Audrey Dorofee, NASA Kennedy Space Center
Larry Druffel, Rational
M. Preston Mullen, Jr., Naval Research Laboratory
Robert Nelson, NASA Goddard Space Flight Center
Edward W. Ng, Jet Propulsion Laboratory

Software Standards Panel:

George Tice, Tektronix, co-chair
Sam Redwine, Institute for Defense Analysis, co-chair
Jerry L. Raveling, Sperry
R. D. Stein, Rocketdyne Div., Rockwell International
David Weiss, Naval Research Laboratory
Bill Wison, NASA Headquarters
John Wolfsberger, NASA Marshall Space Flight Center
Paul B. Wood, Southwest Research Institute

ATTENDEES

NASA MSFC

ADAIR, BILLY
AICHELE, DAVID
BLEVINS, HAROLD R.
BRADFORD, W. C.
BROWN, H.E.
BURRUSS, GLENDA
BUTLER, C.S.
COZELOS, CHARLES
CRAFT, RAY H.
HALEY, SAM
HARDING, GEORGE
HAUFF, C.
HELLMAN, CATHY
HILLIARD, JAMES W.
HINKLE, KENNETH
JERNIGAN, TIM
KYNARD, MIKE
LAMBING, STEVEN J.
LIDE, W.C.
LYNCH, THOMAS J.
MITCHELL, WALTER
MOORE, CARLETON
MORRIS, CHARLES
MULLINS, LARRY
SCOTT, YVETTE S.
STEVENS, BOB
THOMAS, JOE H.
WANG, CAROLINE
WATKINS, JIMMY R.
WHITE, RON
WILLIAMS, ELLEN
WOLFSBERGER, JOHN

NASA GSFC

MCGARRY, F. E.
MERWARTH, PHIL
NELSON, ROBERT
SMITH, GENE

NASA HEADQUARTERS

BENSIMAN, MARC
BISHOP, J.
FANG, AI
HALL, DANA
HORNSTEIN, RHODA
LOFTON, LOUIE R.
WILSON, BILL

NASA JSC

BURKE, LIANA E.
DIXON, DANE
GORMAN, STEVE
HOUSE, SAM
HOWES, DAVE
JOHNSON, ANGIE
PUTNEY, BARBARA
RAINES, GARY
RANEY, JAMES L.

NASA KSC

DOROFEE, AUDREY
PURER, TOM

NASA LARC

JONES, W.R.
VOIGT, SUSAN

NASA LERC

DANIELE, CARL
SOEDER, JAMES

BCSS

TAYLOR, J. H.

BOEING AEROSPACE CO.

BISHOP, KAILE
ELSTON, J. R.
PURVES, BYRON

BOEING COMPUTER SERVICES

CALLAHAN, LOUIS R.
GILL, CHRISTOPHER

CENTURY COMPUTING

ROY, DANIEL
SHAW, CHARLES E. III

CINCOM SYSTEMS, INC.

THORNHILL, ED
TILLOTSON, KATHERINE A.

COMPUTER CORPORATION OF AMERICA

GRAF-WEBSTER, ERIKA

COMPUTER SCIENCES CORP.

AGRESTI, WILLIAM W.
CHURCH, VIC
LANGDON, WOODY
VALLONE, A. DR.

COMPUTER TECHNOLOGY ASSOCIATES

COOLEY, CAROLYN
HEYLIGER, GEORGE
VELEZ, TOM

COMPUTER THOUGHT CORP.

SPRAY, ROB

COMPUTING TECHNOLOGY
TRANSITIONS, INC.

MANLEY, JOHN

COSMIC

GIBSON, JOHN A.

C.S. DRAPER LABS

DEWOLF, J. B.
FELLEMAN, PHILLIP G.
WHALEN, M.

CSI, INC.

SPEIGHT, HOWARD L.
WARTHMAN, JAMES L.

DIGITAL EQUIPMENT CORP.

SOBERA, RICK

ENSO, INC.

MARINE, RALPH W.

EXPERTWARE, INC.

MARCINIAK, JOHN

GENERAL DIGITAL INDUSTRIES, INC.

BOUNDS, RANDY

GENERAL DYNAMICS

MURRAY, WILLIAM M.

GENERAL ELECTRIC

HSIEH, JIM
ZUCKER, SANDRA

GRUMMAN

EVERETT, RALPH
MOONEY, CHARLES S.

GTE COMMUNICATION SYSTEMS CORP.

LAMONTAGNE, G. A.
STAGER, TOM

HARRIS CORP.

BOWMAN, WILLIAM LEE
JENNINGS, WILLIAM S.
LEMARD, TOM

HR TEXTRON

DE FEO, PIO V.

HUGHES AIRCRAFT CO.

COURT, TERRY D.

IBM

DASHIELL, CHERYL
DYER, MICHAEL
GRIM, CLIFTON III
JAKUBOWSKI, JAN
LUI, OSCAR Y.
MEYER, ROBERT W.
MILLS, HARLAN DR.
PARKER, CONNIE
SPOTZ, WILLIAM H.

ICASE

NOONAN, ROBERT DR.

IMPAR, INC.

KIDD, J.
PARKER, R.R.

INSTITUTE FOR DEFENSE ANALYSIS

REDWINE, SAM

INTERGRAPH CORP.

ECK, CLARENCE

INTERMETRICS, INC.

KIRCHOFF, MARJORIE
MOOREHEAD, DELORES S.
MURRAY, SANDRA K.
WALKER, DAVID L.

ISDOS, INC.

TEICHROW, DANIEL DR.

JPL

CALLENDER, DAVID
GIFFIN, GEOF
MAUND, DON
MCKENZIE, MERLE
MCLEOD, JOHN
NG, EDWARD
STEINBACHER, JODY

KMG/MAIN HURDMAN

MURPHY, D. PAUL

LOCKHEED

ALLEY, D.J.
ELY, EDWARD
KELLEY, P.B.
RATLIFF, ALAN W.
WILKINSON, JOHN

MARTIN MARIETTA AEROSPACE CORP.

LABAUGH, ROBERT
PERRONE, GIOVANNI
SEYMOUR, HENRY

MCDONNELL DOUGLAS CORP.

LOKKEN, WILLIAM R.
MCCABE, JOE
MONTOYA, GONZALO

MDTS CO.

CONSTANTINE, RANDOLPH

MITRE CORP.

CHRISTOPH, BARBARA A.
DE LONG, CLYDE S. JR.
JACKSON, R. M.
KAUTZMAN, FRANK DR.
LORENTZ, JEFFREY L.
LOXTON, JOHN M.

NATIONAL BUREAU OF STANDARDS

HANKINSON, AL

NAVAL POSTGRADUATE SCHOOL

SCHNEIDWIND, NORMAN

NAVAL RESEARCH LAB.

FAULK, STUART R.
MULLEN, M. PRESTON, JR.
WEISS, DAVID DR.

NUCLEAR STRUCTURES, INC.

GENSER, JOHN ROBERT

ORI, INC.

BURNS, RICHARD W.

PERKIN-ELMER CORP.

FITZGERALD, A. J.

RATIONAL COMPUTING SYSTEMS

DRUFFEL, LARRY DR.
GORDON, DEBRA

REIFER CONSULTANTS, INC.

REIFER, DON

RESEARCH TRIANGLE INSTITUTE

DUNHAM, JANET R.

ROCKWELL INTERNATIONAL

BROWN, REX B.
HALL, RONALD O.
LEONARD, E.L.
SINCLAIR, CRAIG C.
STEIN, R. D.
VRIELING, R. I.

SCIENCE APPLICATIONS INTERNATIONAL

BROWN, JULIA A.
BURSON, DICK
FAGUE, MIKE
HORNER, JACK H.
JONES, CARL
LOVELESS, ROBERT L.
QUINN, MARK T.

SDC

MILLAR, FRANK
MUNSON, JOHN (JACK) B.
PAYTON, TERI

SINGER CO.

SEBASTIAN, WILLIAM G.
SIVILLO, PETE

SOFTECH

AUTY, DAVE
FISHBEIN, SEYMOUR
KESSINGER, RICHARD

SOFTWARE QUALITY ENGINEERING

GELPERIN, DAVID

SOUTHWEST RESEARCH INSTITUTE

WOOD, PAUL B.

SPERRY

MCCLURG, JIM
MEIRINK, MICHAEL J.
RAVELING, JERRY
WINCH, DARREL

STANFORD TELECOMMUNICATIONS CORP.

ZAKRZEWSKI, EDWIN

SUTRON CORP.

SLAGER, RANDY

SYSTEM TECHNOLOGY INSTITUTE

LOESH, BOB
SANSON, MICHAEL

TEKTRONIX, INC.

SENDOWSKI, DAVID
TICE, GEORGE D.,JR.

TELEDYNE BROWN ENGINEERING

BEATY, RON
CLYMER, SUZANNA J.

TRW

BOEHM, BARRY DR.
BRASLAU, ROBERT
GUILLEBEAU, M.
HARRIS, STEVEN
HOLLOWICH, MIKE
IRBY, JOHN
MCALISTER, ED
NORMAN, GEORGE
PHILLIPS, DAVE
STUCKLE, DON
WALKER, DAVID C.
WHEELER, SUZAN
WOOD, LOREN

VERDIX CORP.

DELLER, STEVEN

UNIVERSITY OF ALABAMA A&M

SAHA, H. DR.

UNIVERSITY OF CALIFORNIA

TAYLOR, RICHARD

UNIVERSITY OF HOUSTON

MCKAY, CHARLES

UNIVERSITY OF MARYLAND

BASILI, V. DR.
ZELKOWITZ, MARVIN

UNIVERSITY OF MICHIGAN

MUDGE, TREVOR DR.

UNIVERSITY OF NEW MEXICO STATE

KNOEBEL, ROBERT A.

UNIVERSITY OF VIRGINIA

KNIGHT, JOHN DR.

OAKWOOD COLLEGE

GILL, ESTHER

FOREIGN PARTICIPANTS

CANADA/NRCC

BOXALL, HARRY

JAPAN/NASDA

IWASAKI, N.
KATO, T.
SUZUKI, IWAO

NETHERLANDS/ESA/ESTEC

ALLEN, R.C.
RENAI, RENARDS
ROBINSON, P.J.
STEVENS, RICHARD

EXECUTIVE SUMMARY

The Open Forum on Space Station Software Issues, held at Marshall Space Flight Center on April 24-25, 1985, was sponsored by the Space Station Program Office and organized by the Space Station Software Working Group. Participation was limited to those willing to submit position statements and each invitation included a copy of "Space Station Software Issues" (NASA CP-2361, 1985). About 225 participants from industry, government, universities, and a few foreign space agencies attended. Four panels, consisting of invited experts and a few NASA representatives, focused on the following topics: (1) software management, (2) software development environment, (3) languages, and (4) software standards.

The forum began with an overview of the Space Station program and some major software issues to be addressed. Then four invited experts spoke on the panel topics, critiquing the strategies outlined in the proceedings of the previous NASA workshop (NASA CP-2361). These talks provided the starting point for the panel discussions which followed.

Each panel deliberated in private and also held two open sessions with audience participation. Major recommendations to the NASA Space Station Program developed by these panels are summarized below.

1. The software management plan should establish policies for software acquisition (treating internal development as if it were external acquisition) and should clearly address responsibilities and decision points. Software should be treated as part of the overall system engineering and integration effort.

2. NASA should furnish a uniform, modular software support environment (with a layered architecture) and require its use for all Space Station software acquired (or developed). This environment should be incrementally developed, have a virtual operating system, and support portable software packages.

3. The language Ada should be selected now as the primary source language for Space Station software, and NASA should begin to address issues related to the effective use of Ada, such as education, a transition strategy, run-time support, and accommodating the use of existing software. Languages for special applications such as requirements analysis and user interface should be selected soon after the requirements become more clear.

4. The Space Station Program should endorse and support software standards through policy and implementing organizations. Selected standards should be tailored for Space Station, based upon existing NASA, DoD, IEEE, and ANSI standards.

Some common themes were expressed by the panels and many audience participants: Do not reinvent the wheel (learn from past experience), obtain "real" requirements for software support, identify and manage interfaces early, focus on maintenance as a primary requirement, include software as an integral part of the system level strategy, and define terminology ("buzz words").

INTRODUCTION

This report summarizes the results of the Open Forum on Space Station Software Issues, held at the NASA Marshall Space Flight Center on April 24-25, 1985. This forum was sponsored by the Space Station Program and organized by the Space Station Software Working Group (SWWG). An earlier workshop on Space Station Software Issues had been held for NASA software specialists in August 1984 (documented in ref. 1). This forum was organized as a follow-up to that workshop to solicit comments from industry and academic representatives on the NASA perspective on software policies and strategies for the Space Station Program.

The objectives of the forum were to define and review major Space Station software issues that should be of concern to NASA, and propose policies, proce- cedures, and actions that should be taken by NASA to effectively address these issues.

The Software Working Group has grown from an ad hoc group in 1983 to an inter-center NASA committee of over 30 members, with a common concern that software issues be addressed early in the Space Station Program definition and development. The workshop held for the SWWG members in August 1984 identified over 20 software issues and made recommendations for each. Since that time, several developments have occurred:

1. Identification of Software Managers in the Space Station Program at Levels A and B, namely Dana Hall and M. Dane Dixon.

2. Drafting of the top-most software management plan.

3. Establishment of Space Station Information Systems Panel responsible for all software systems.

4. Drafting of Space Station Software Lexicon.

5. Plans made for software development environment and Space Station language evaluation.

The forum was organized to present the NASA perspective on the role of software in Space Station (given by Dana Hall) as well as four expert views on the NASA proposed approach to software management, software development environment, languages, and software standards. Following the expert presentations, four panels met to deliberate on these topics and to hold open sessions inviting comments from the audience. Each panel met once in closed session and twice with members of the forum audience. They then presented their tentative recommendations at the final forum session. After the meeting, the panel members refined their statement of issues and recommendations and these are presented in this document.

Note that the recommendations presented herein have not been endorsed by the NASA Space Station Program; however, they are receiving careful consideration by NASA officials at the time of publication of this report.

D1

N86-23315

SPACE STATION - THE ROLE OF SOFTWARE

Dana Hall*
NASA Office of Space Station
Washington, DC

ABSTRACT

Software will play a critical role throughout the Space Station Program. This presentation is intended to set the stage and prompt participant interaction at the Software Issues Forum. The presentation is structured into three major topics:

- an overview of the concept and status of the Space Station Program;
- several charts designed to lay out the scope and role of software;
- and information addressing the four specific areas selected for focus at the forum, specifically: software management, the software development environment, languages, and standards. The presentation attempted to highlight NASA's current thinking and to raise some of the relevant critical issues.

*Dr. Dana Hall is the Level A Space Station software manager and is responsible for oversight of the planning, implementation, and integration of all Space Station Program software. Prior to joining the Program in October 1984, Dr. Hall served as a data system and software advisor within NASA's Office of the Chief Engineer. His prior experience is with MITRE and TRW where he has worked with projects ranging from airline operations models to missile trajectory simulations. Dr. Hall has also been involved with NASA data system advanced development and in the ground system design of several NASA spaceflight programs.

~~ORIGINAL PAGE COLOR PHOTOGRAPH~~

N86-23315

NEXT LOGICAL STEP

~~ORIGINAL PAGE COLOR PHOTOGRAPH~~

Given the advent of an operational space transportation system, the Space Shuttle, the development of a space station is the next logical step in mankind's exploration of the surrounding universe.

ORIGINAL PAGE IS OF POOR QUALITY

STATE OF UNION

The Space Station Program traces its official beginning to the January 1984 State of the Union message by President Reagan in which he directed that NASA proceed to develop a "permanently manned space station and do it within a decade." This official start builds upon many years of prior analyses and considerations that together laid the basic guidelines that now comprise the Space Station Program.

"We can follow our dreams to distant stars, living and working in space for peaceful, economic and scientific gain. Tonight, I am directing NASA to develop a permanently manned space station and to do it within a decade.

A space station will permit quantum leaps in our research in science, communications and in metals and life-saving medicines which can be manufactured ... in space."

January 25, 1984

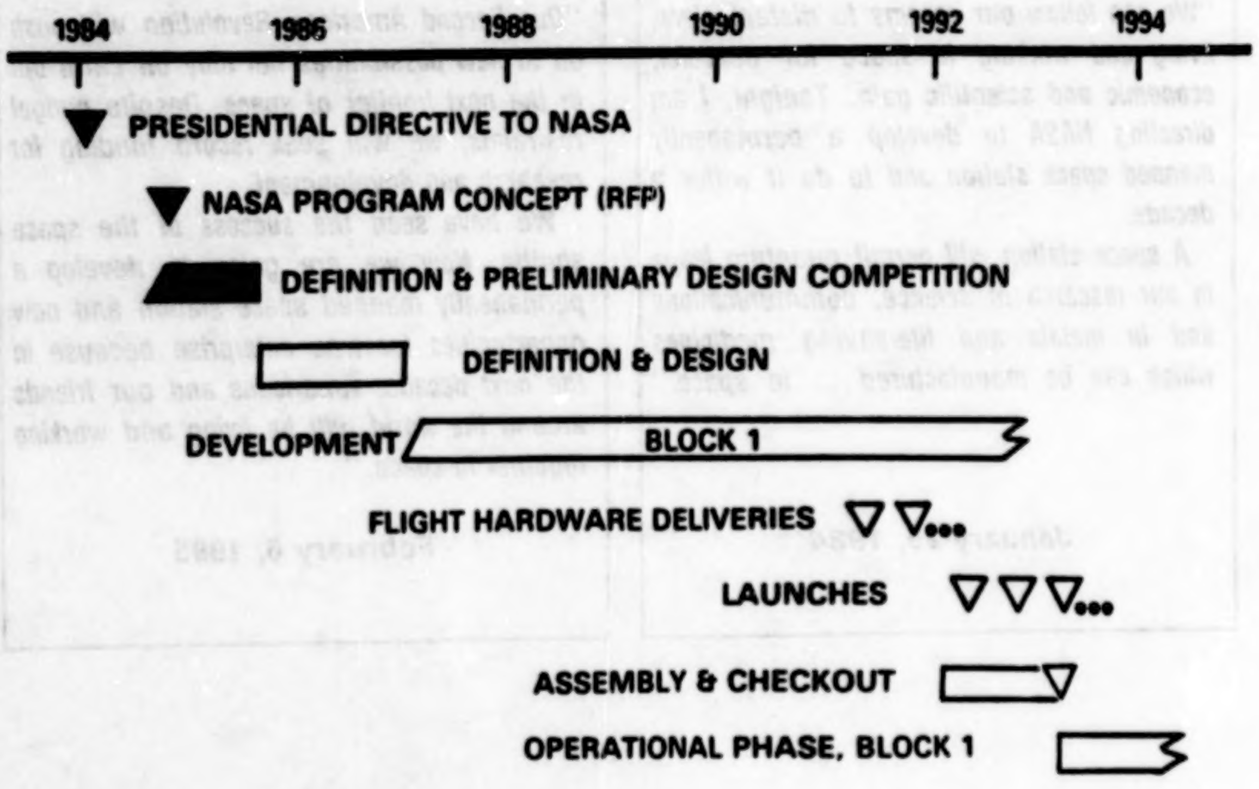
"Our Second American Revolution will push on to new possibilities not only on Earth but in the next frontier of space. Despite budget restraints, we will seek record funding for research and development.

We have seen the success of the space shuttle. Now we are going to develop a permanently manned space station and new opportunities for free enterprise because in the next decade, Americans and our friends around the world will be living and working together in space."

February 6, 1985

MILESTONES

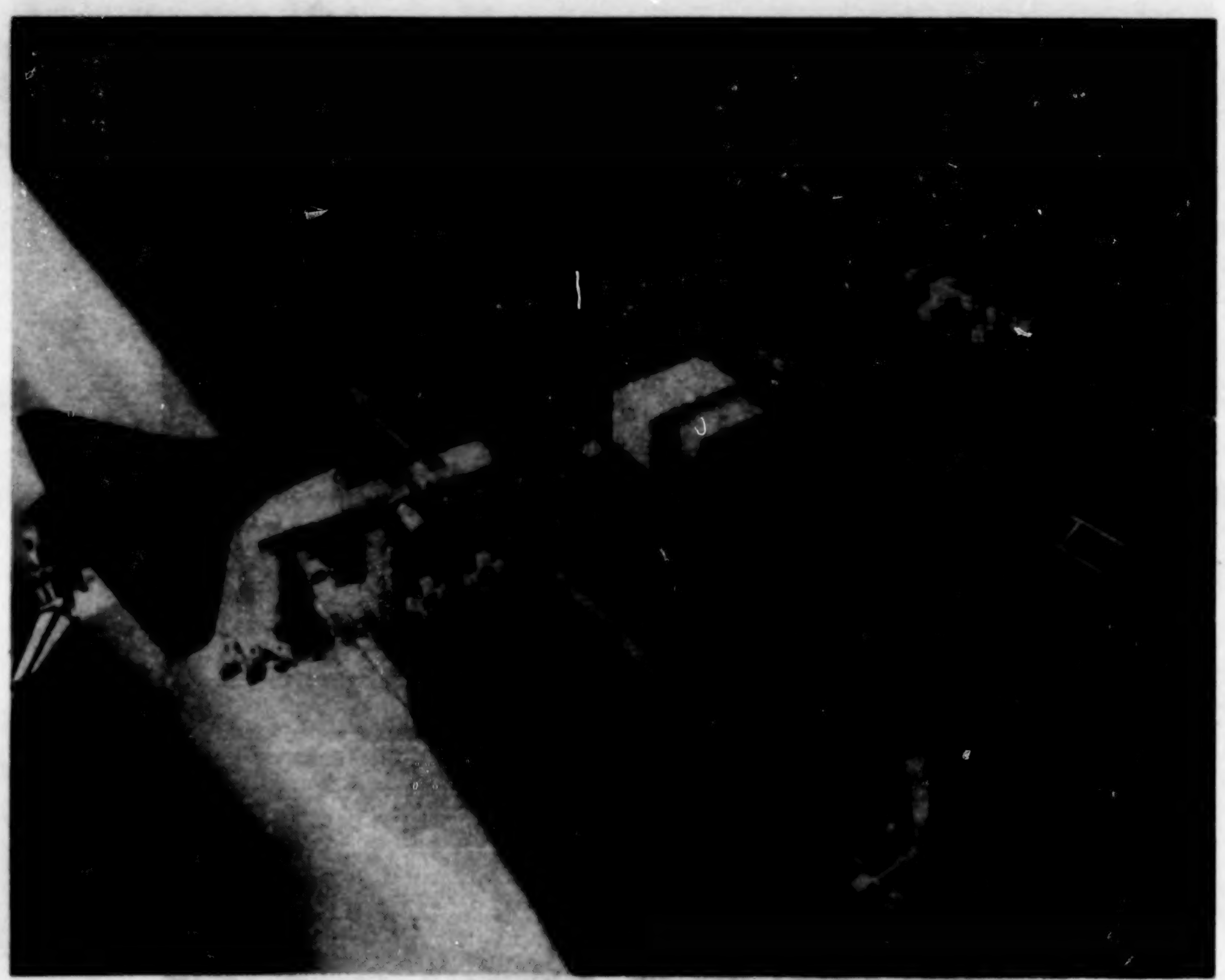
At the time of this forum, the program had just completed the competition for the definition and preliminary design of the Space Station elements. This competition resulted in the award of eight major contracts distributed across four primary work packages. As shown on this schedule, it is planned that actual development (i.e., Phase C/D) will begin in 1987. Initial operational capability is forecast for the 1993-94 time frame.



ORIGINAL PAGE IS
OF POOR QUALITY

SPACE STATION DESIGN

The actual design of the Space Station is not known at present since the program is still in the requirements and definition part of its life cycle. However, NASA had adopted a reference configuration, as shown in this artist's concept.

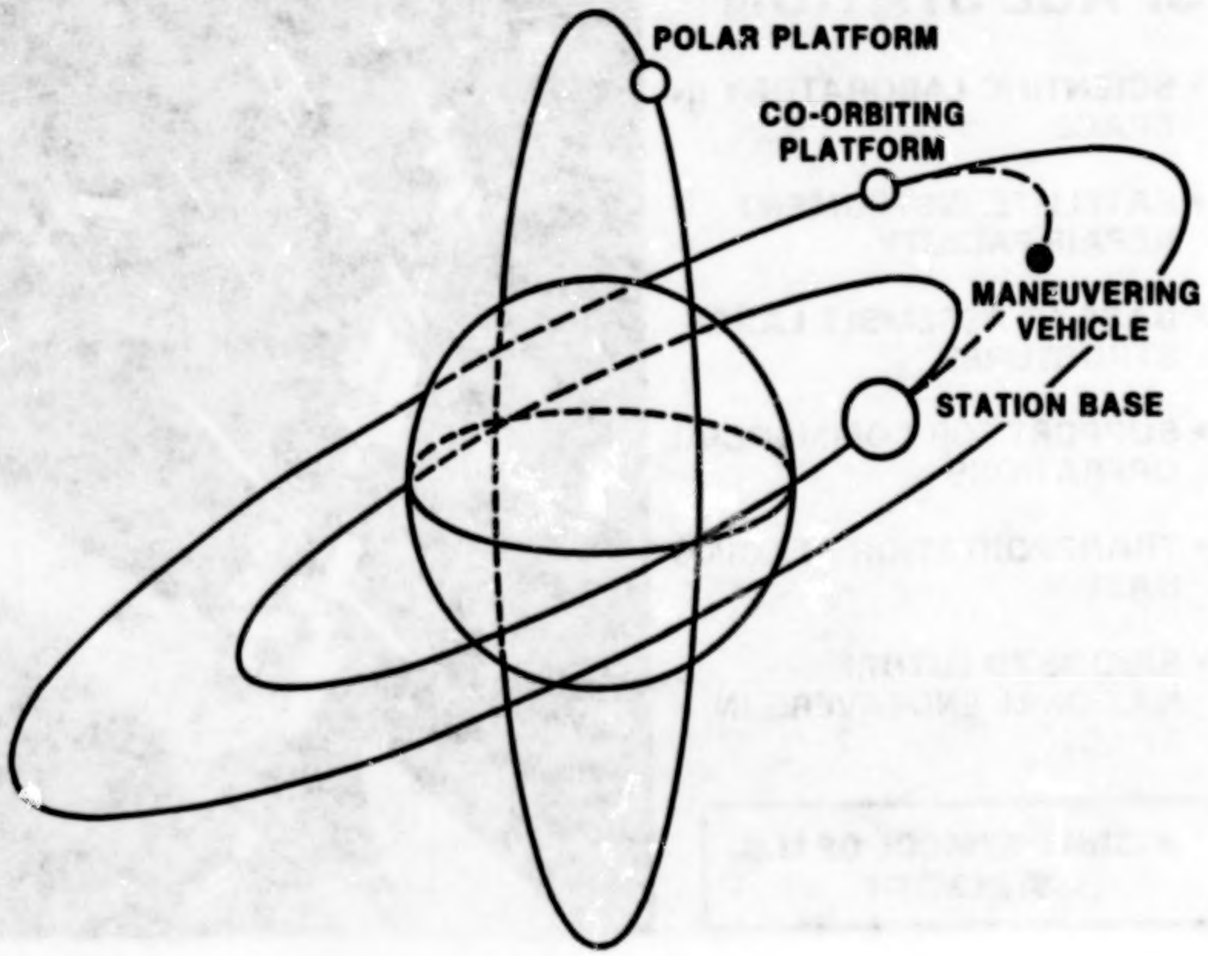


REFERENCE CONFIGURATION

As shown, the reference configuration is an elongated truss-like structure approximately 400 feet long and 200 feet wide. It will be maintained in a 250 n.mi. circular orbit inclined at 28.5 degrees to the equator. The station will be oriented in a gravity gradient attitude with Earth sensing payloads and the various modules located on the end closest to the Earth. The present concept is that the station will be powered by solar arrays. Also shown are two orbital maneuvering vehicles. These OMVs will be unmanned, remotely controlled spacecraft designed to ferry payloads and equipment in nearby ranges. One such destination might be a co-orbiting unmanned platform, as shown on the sketch.

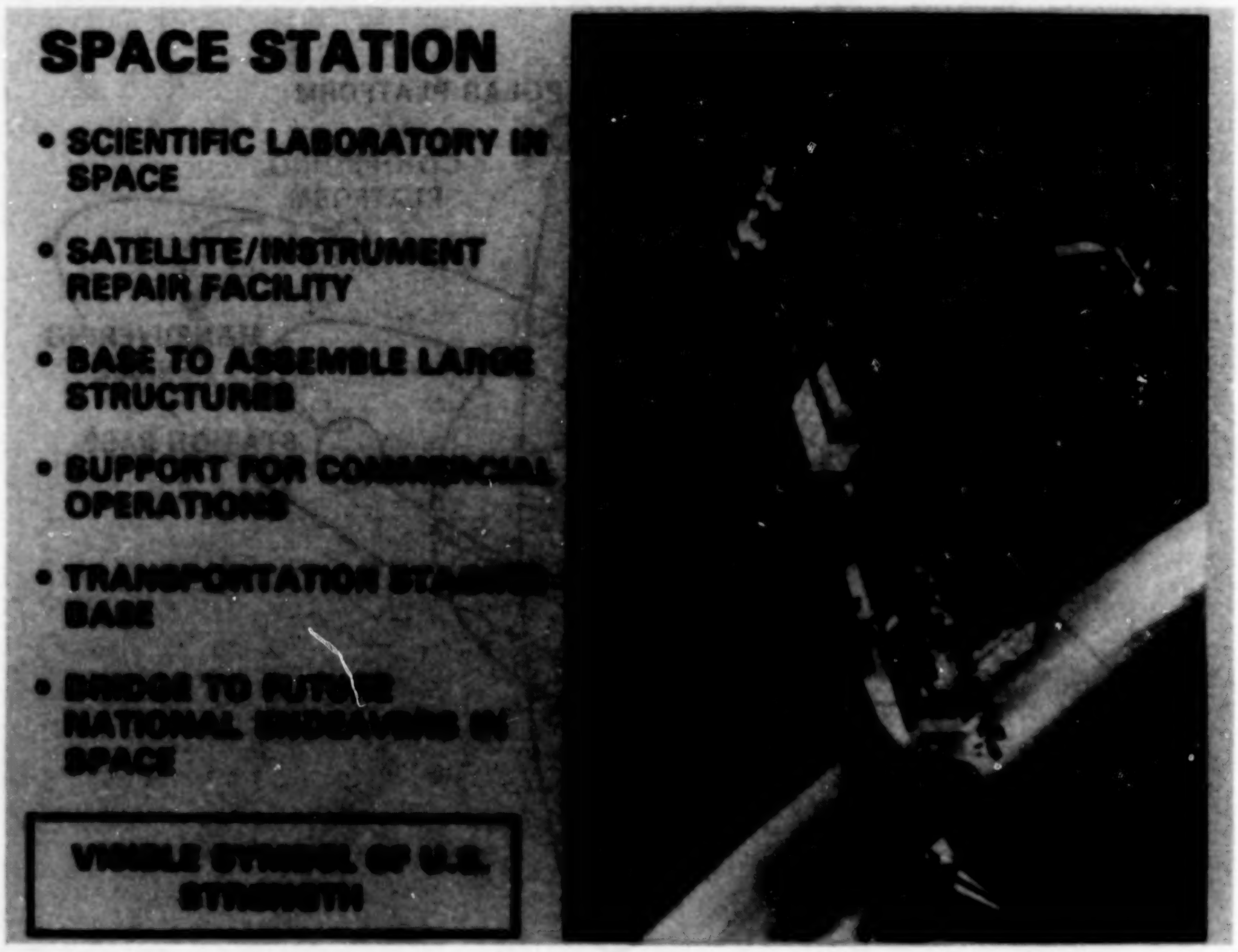
SPACE STATION COMPLEX

The Space Station Complex consists of three major elements. Two of those elements are the Space Station Main Base, discussed in the previous figure, and, in that same 28.5 degree orbit, an unmanned platform. The third major element of the Space Station Complex is an unmanned Polar Platform. The Polar Platform will be the location for most Earth sensing instruments since that platform will survey all of the Earth's surface on a frequent basis. The figure also shows one of the orbital maneuvering vehicles traveling between the Space Station Main Base and the Co-Orbiting Platform.



SPACE STATION

The Space Station will serve as a means for furthering our scientific research in space and will have a number of additional important functions. One will be as a satellite or instrument repair facility, a capability that has been demonstrated using the Shuttle Program. Space Station will also serve as a base to assemble large structures. It will be a facility to support the commercialization of space and a transportation staging base for missions to the Moon and beyond. Overall, the Space Station will be a visible symbol of U. S. strength.



SPACE STATION

- SCIENTIFIC LABORATORY IN SPACE
- SATELLITE/INSTRUMENT REPAIR FACILITY
- BASE TO ASSEMBLE LARGE STRUCTURES
- SUPPORT FOR COMMERCIAL OPERATIONS
- TRANSPORTATION STAGING BASE
- BRIDGE TO FUTURE NATIONAL ENDEAVORS IN SPACE

VISIBLE SYMBOL OF U.S. STRENGTH

SPACE STATION PLANNING GUIDELINES

A number of management and engineering guidelines have been established for the Space Station Program. The management guidelines include provisions for an initial operational capability station within a decade. The program has very extensive user involvement both from our traditional communities of the scientific and application areas as well as from technology and from the commercial sector. On the technical side, the station must be evolutionary in nature and technology transparent. We are looking at a Space Station Program with a lifetime of something like 25 to 30 years and thus must be able to change our technology without impacting the users. The station elements will be serviced by the Shuttle. The Space Station Main Base will be continuously habitable.

MANAGEMENT RELATED

- **Three year detailed definition (5-10% of program cost)**
- **NASA-wide participation**
- **Development funding in FY 1987**
- **IOC: "within a decade"**
- **Cost of initial capability: $8.0B**
- **Extensive user involvement**
  - **— Science and applications**
  - **— Technology**
  - **— Commercial**
- **International participation**

ENGINEERING RELATED

- **Continuously habitable**
- **Shuttle dependent**
- **Manned and unmanned elements**
- **Evolutionary**
- **Maintainable/restorable**
- **Operationally semi-autonomous**
- **Customer friendly**
- **Technology transparent**

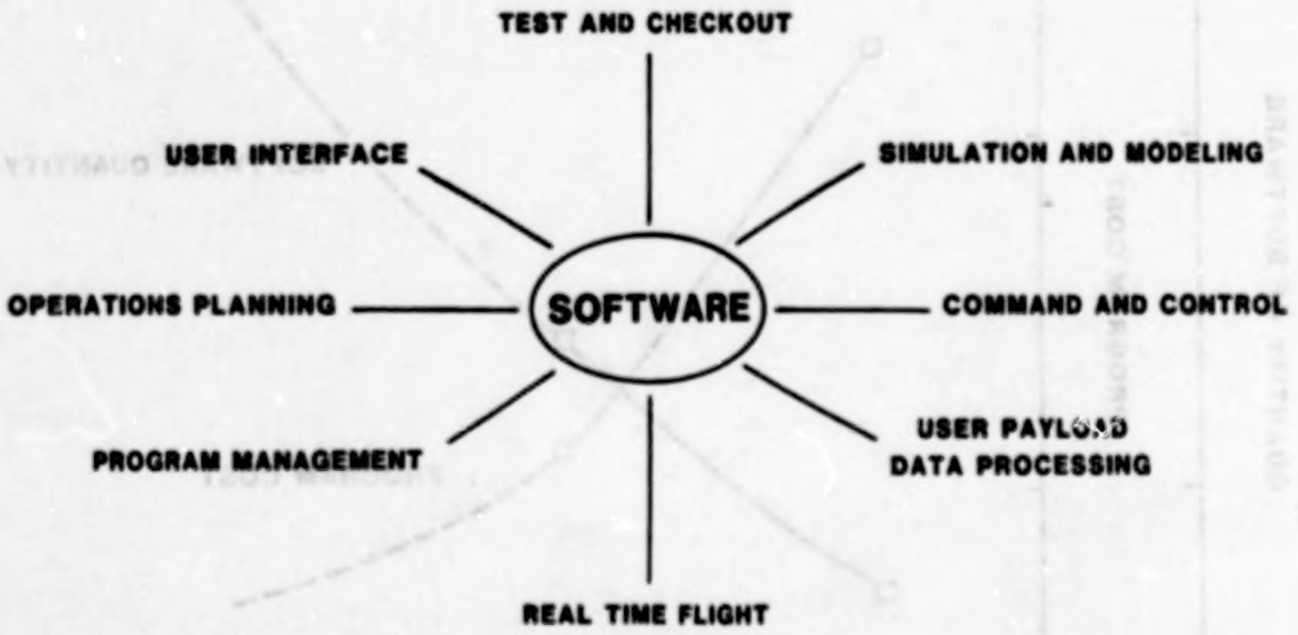
INTERNATIONAL COOPERATION

President Reagan as part of the Space Station initiation invited international participation. We are pleased to welcome the European Space Agency, Canada, and Japan to our team. The Memoranda of Understanding between ourselves and those participants are soon to be signed.

- **PRESIDENT REAGAN INVITED INTERNATIONAL PARTICIPATION**
- **ESA, CANADA AND JAPAN HAVE RESPONDED:**
  - **SOON TO SIGH MOU'S ON PHASE B COOPERATION**
- **SPACE STATION IS TO BE A TRULY COOPERATIVE ENDEAVOR:**
  - **DEVELOPMENT**
  - **UTILIZATION**
  - **OPERATIONS**
- **U.S. AND FOREIGN INDUSTRIES MAY COOPERATE TOO**

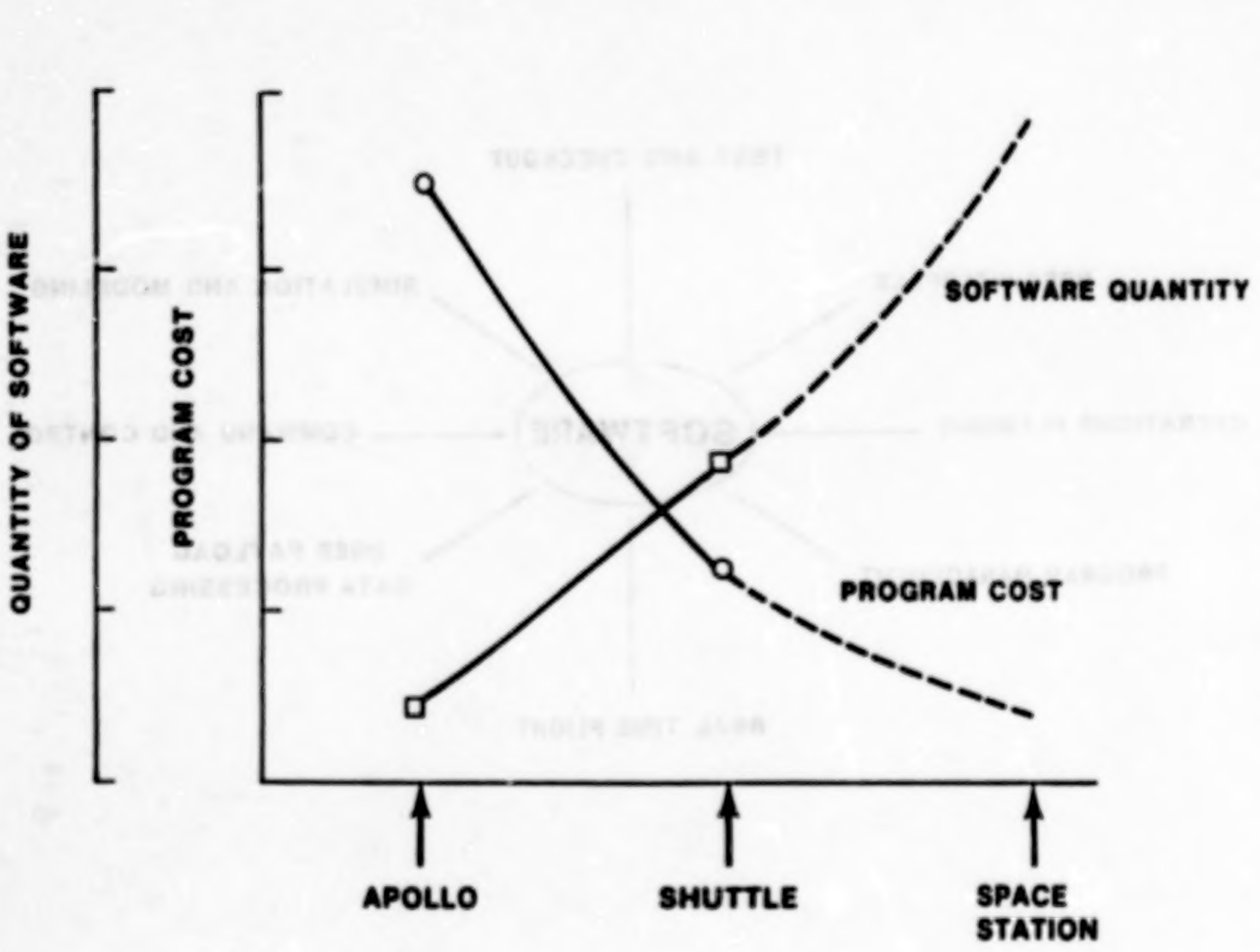
PROGRAM SUCCESS

Software will play a very critical role throughout the Space Station Program. This figure illustrates just a few of those major categories. They range from test and checkout to user interface support, payload processing, command and control, and of course management of the program itself.



NASA SOFTWARE TRENDS

This figure tries to show in an unquantified manner the significant increase in software that we believe we will be working with in the Space Station Program compared to the amount that we developed for Apollo and Shuttle. It also shows that the Space Station effort will be built with substantially less dollars than were available on those past major programs. So the primary messages from this and the previous figure are that NASA must maximize the efficiency with which it uses its software resources. We must learn as much as we can from past lessons, be careful not to repeat mistakes, and use methodologies that worked well before.



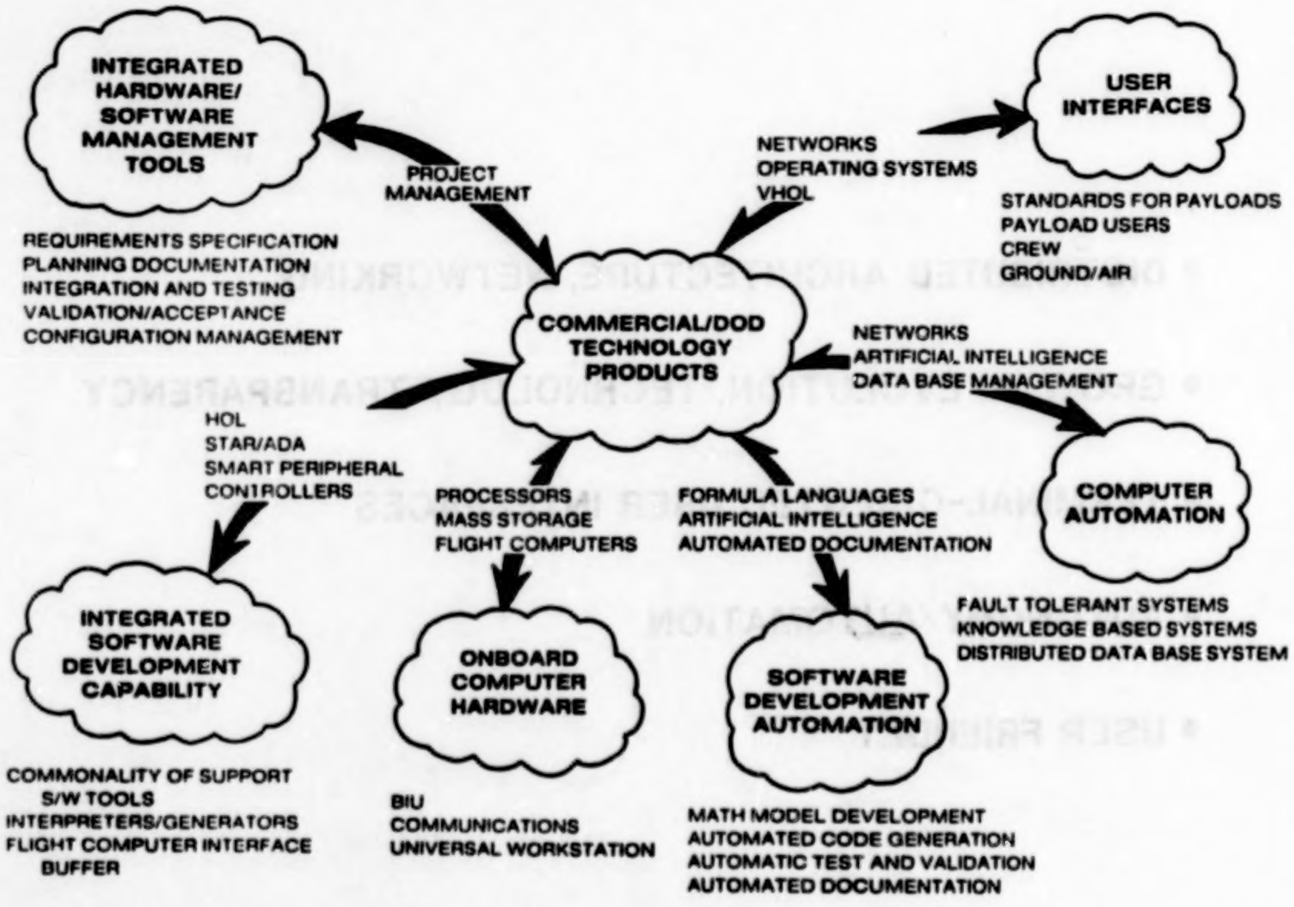
FUNDAMENTAL REQUIREMENTS

This figure lists a few of the major requirements that are software drivers. We recognize that we will be working with a highly distributed architecture and that networking will be prevalent throughout that architecture in the form of local area networks as well as wide area networks. As we said earlier, the station technology on-board and on the ground must be oriented for growth and evolution. Our users will be working from terminals via a space station information system that we plan will enable those users to operate just as if their instruments were in the laboratory next door. The Space Station will at least initially have a crew of somewhere between six to eight and therefore automation will be important. Many of the functions on-board the station must perform in an autonomous manner and since we are looking at a long term program, we must try to automate as much of the ground system as we can to minimize operating costs. Of course, the overall driving requirement is that the entire system be user friendly both for NASA operators running the station and for our customers.

- **DISTRIBUTED ARCHITECTURE, NETWORKING**
- **GROWTH, EVOLUTION, TECHNOLOGY TRANSPARENCY**
- **TERMINAL-ORIENTED USER INTERFACES**
- **AUTONOMY/AUTOMATION**
- **USER FRIENDLY**

TECHNOLOGY

There are a large number of commercial and Department of Defense technology products that can potentially be used to serve all of the areas on this figure. These include integrated hardware and software tools, on-board computer hardware, software development aids, computer automation, and aids for the user interface. Notice that the arrows go two ways. The two-direction arrows show that in some cases some of what NASA does with these products may influence the commercial and DOD sectors. However, that is not the main message. The bottom line is that we plan to maximize the use of commercial and DOD products.



SOFTWARE MANAGEMENT

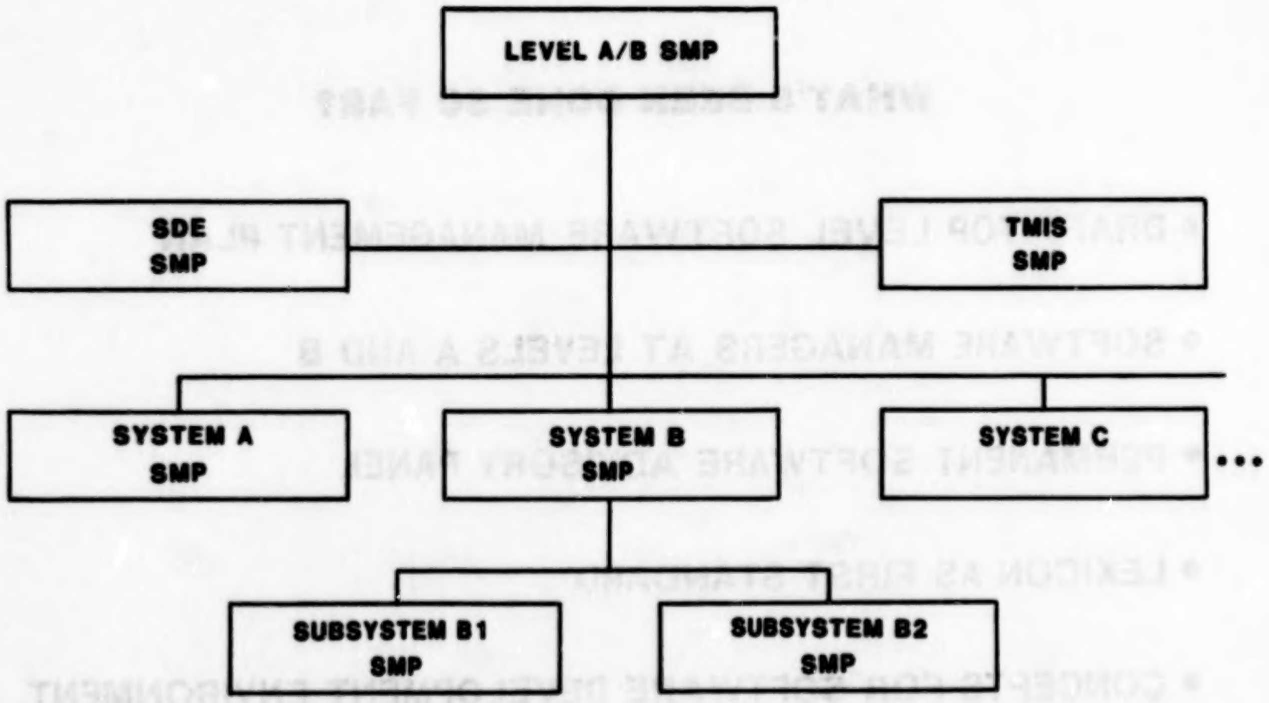
NASA has taken a number of software management steps that we think are positive and in the right direction. First of all, the top level (combined Levels A and B) software management plan has been drafted. That document will continue throughout the program's life to be the repository for the program's policies and procedures. We have also created positions and appointed people as designated software managers at Levels A and B. The Program is in the process of converting what has been an ad hoc software working group into a permanent software advisory panel. We are beginning to assemble software standards, the first of which will be a lexicon so that all participants will be using the same defintion of terms. And finally, we are in the latter stages of conceptualizing a software development environment. Now let me pause for a moment here and clarify that we also refer to the SDE as a software support environment, the idea being that the term "support" conveys a wider process than does development. We presently use both terms synonymously.

WHAT'S BEEN DONE SO FAR?

- DRAFT TOP LEVEL SOFTWARE MANAGEMENT PLAN
- SOFTWARE MANAGERS AT LEVELS A AND B
- PERMANENT SOFTWARE ADVISORY PANEL
- LEXICON AS FIRST STANDARD
- CONCEPTS FOR SOFTWARE DEVELOPMENT ENVIRONMENT

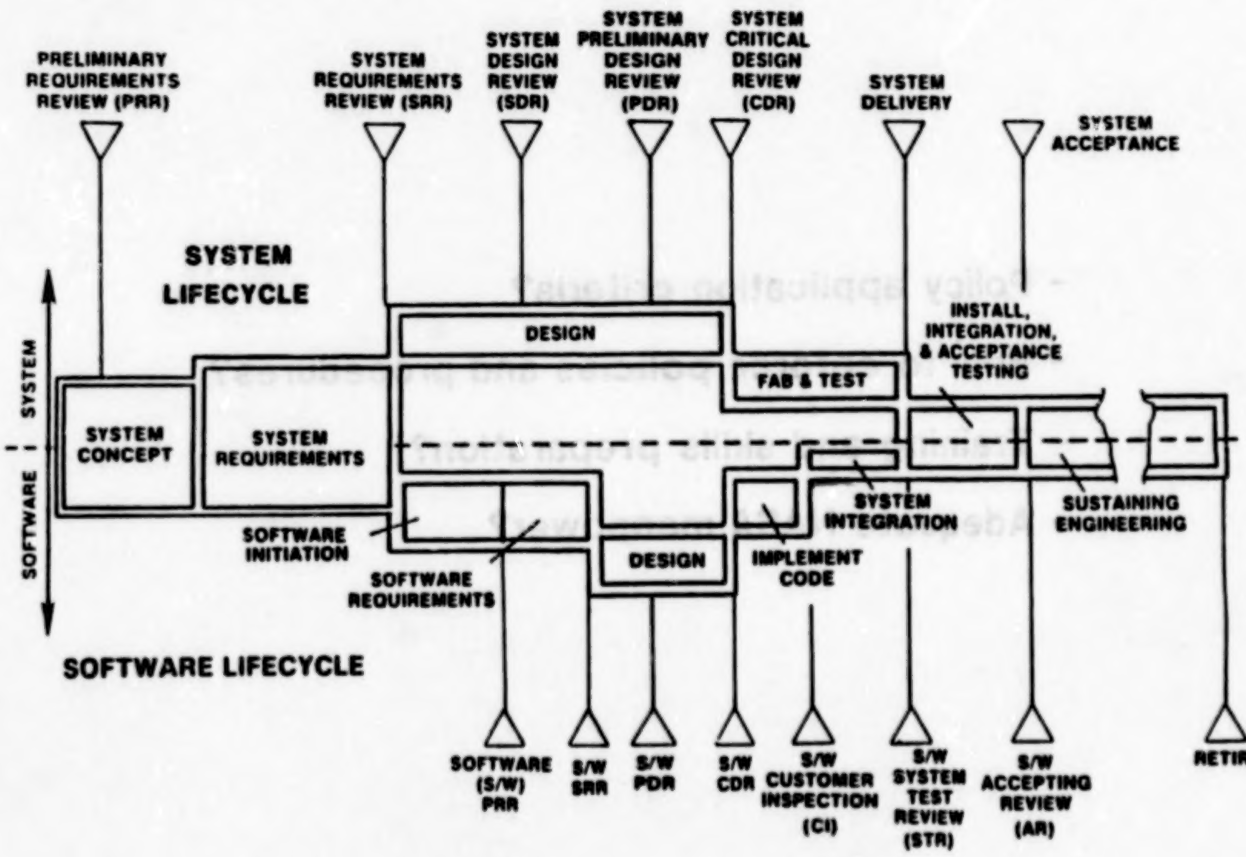
SOFTWARE MANAGEMENT PLANS

The Space Station Program envisions a hierarchy of software management plans, i.e., one plan per major software element. The plan at the top of this figure, the Level A/B software management plan, is the one that is presently in draft form and that will soon be undergoing formal review throughout the Space Station Program. Two other major elements that have been identified so far will also be required to have individual software management plans. One is the software development environment (or the software support environment) and the other is the Technical and Management Information System (TMIS). Since the other elements of the Space Station Program have not yet been identified (we are still in the requirement stage) they are shown on this chart simply as systems A, B, C, and so on.



SPACE STATION LIFE CYCLE

Pictured here is the standard Space Station System and Software Life Cycles that will be used within the Program. The top half of the figure shows what the systems phases are, and the bottom half gives the corresponding software phases. Shown as well are the major reviews and events that will take place across that life cycle. We will require that all space station software efforts utilize the life cycle regardless of whether the software is being developed or acquired. (Ed. note: This life cycle has been slightly modified in the approved Software Management Plan, which will be available from the Space Station Program Office in late 1985.)



PRELIMINARY REQUIREMENTS REVIEW (PRR)
SYSTEM REQUIREMENTS REVIEW (SRR)
SYSTEM DESIGN REVIEW (SDR)
SYSTEM PRELIMINARY DESIGN REVIEW (PDR)
SYSTEM CRITICAL DESIGN REVIEW (CDR)
SYSTEM DELIVERY
SYSTEM ACCEPTANCE
SYSTEM LIFECYCLE
SYSTEM
SOFTWARE
DESIGN
INSTALL, INTEGRATION, & ACCEPTANCE TESTING
FAB & TEST
SYSTEM CONCEPT
SYSTEM REQUIREMENTS
SYSTEM INTEGRATION
SUSTAINING ENGINEERING
SOFTWARE INITIATION
SOFTWARE REQUIREMENTS
DESIGN
IMPLEMENT CODE
SOFTWARE LIFECYCLE
SOFTWARE (S/W) PRR
S/W SRR
S/W PDR
S/W CDR
S/W CUSTOMER INSPECTION (CI)
S/W SYSTEM TEST REVIEW (STR)
S/W ACCEPTING REVIEW (AR)
RETIRE

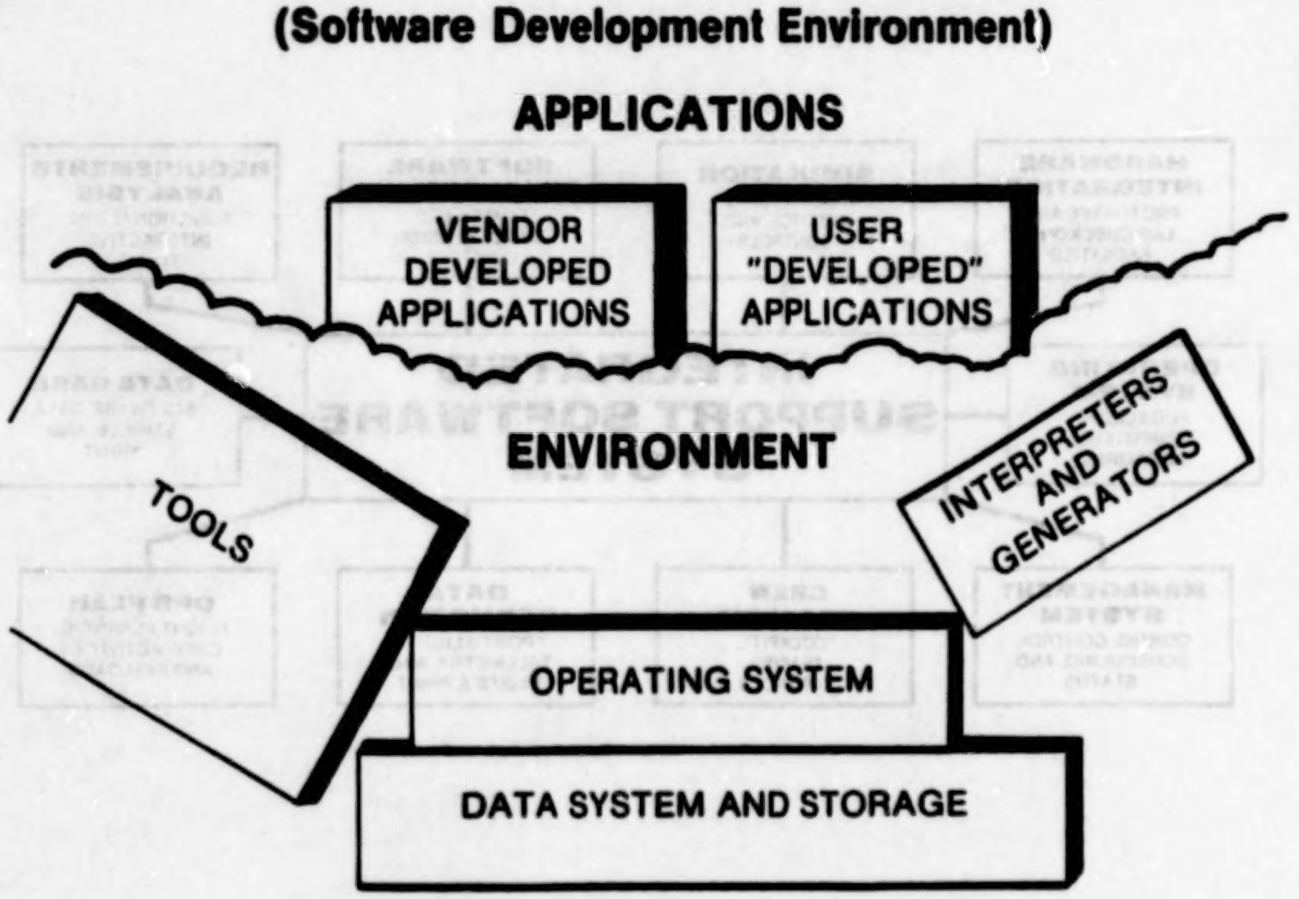
EXAMPLES ISSUES

Although a number of steps have been taken, many additional software management issues remain. This figure lists a few of them. One such issue concerns application criteria, i.e., to what depth and to what extent should our policies and procedures apply? We certainly don't want to impose all these rules on the technical person working in an office with a personal computer. By what criteria do we decide how much of the policies and procedures apply to each element and each situation? Another issue, and a very important one, is how do we enforce these policies and procedures as well as the supporting standards that serve to implement the policies and procedures? What enforcement mechanism should be used? A third issue is training and skills preparation. Is it adequate simply to send our people to management courses, or is additional preparation needed? Should we consider staff rotation into different jobs? A fourth issue is the question of whether NASA has adequate manpower to do the system and software engineering and integration job, and if the answer is no, then what should NASA do?

- **Policy application criteria?**
- **How to enforce policies and procedures?**
- **Training and skills preparation?**
- **Adequate NASA manpower?**

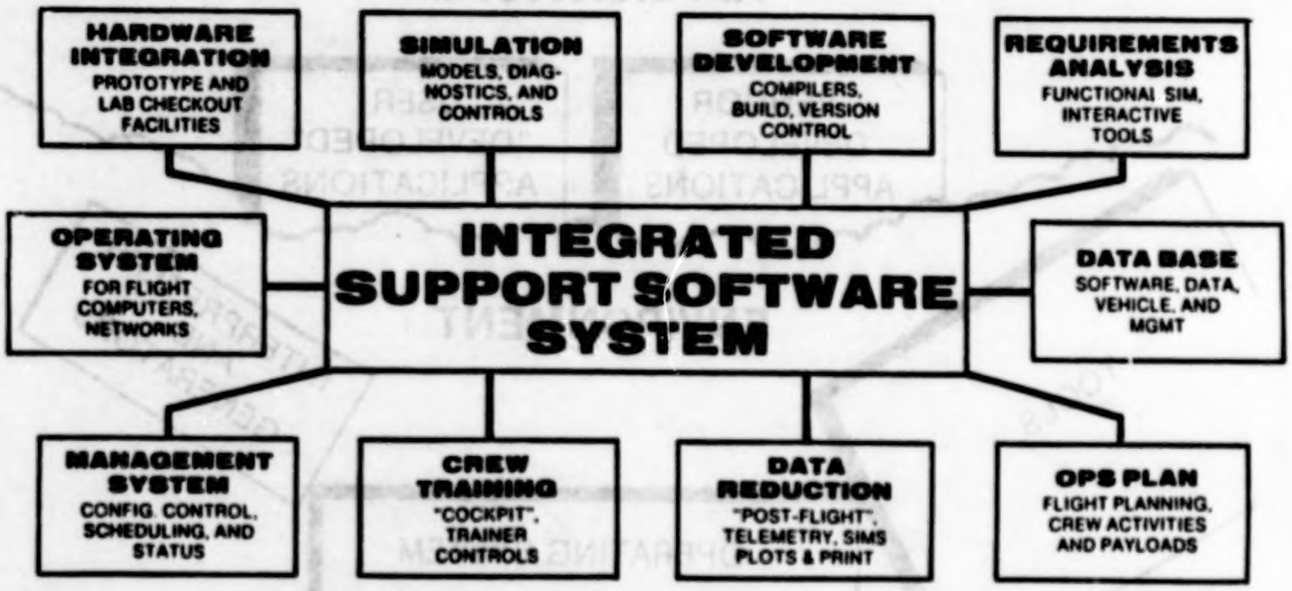
APPLICATIONS ENVIRONMENT

This figure illustrates a couple of important features about the software development environment. One is that the software development environment will consist of a standard set of tools, software packages, policies, and procedures which from one perspective will free the user and the application software from the operating system and data storage. Another important message from this figure is that the user and the application software will be provided a number of services by the software development environment via the tools, interpreters, code generators, operating system, etc., that comprise that software development environment.


(Software Development Environment)
APPLICATIONS
VENDOR DEVELOPED APPLICATIONS
USER "DEVELOPED" APPLICATIONS
ENVIRONMENT
TOOLS
INTERPRETERS AND GENERATORS
OPERATING SYSTEM
DATA SYSTEM AND STORAGE

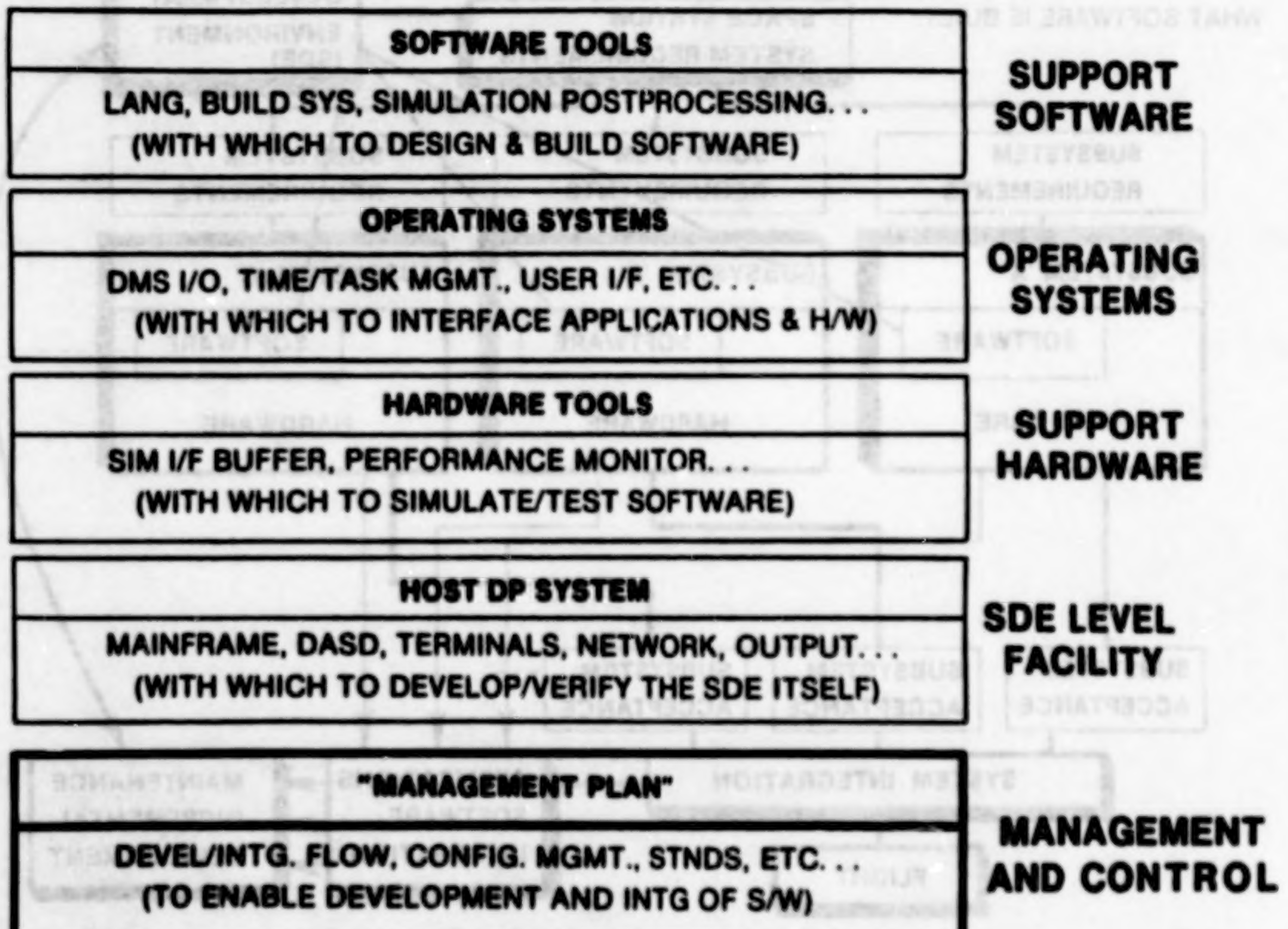
COMMONALITY

This figure illustrates that an integrated software support system will consist of many different elements: aids for hardware integration, simulation models, diagnostics, control tools, software development aids, compilers, version control tools, packages to analyze requirements, operating systems, management systems, and so on. It is also important to realize that the integrated software support system will enable the user to select the particular support elements required and thus form a specific subset software support environment.


HARDWARE INTEGRATION
PROTOTYPE AND LAB CHECKOUT FACILITIES
SIMULATION
MODELS, DIAGNOSTICS, AND CONTROLS
SOFTWARE DEVELOPMENT
COMPILERS, BUILD, VERSION CONTROL
REQUIREMENTS ANALYSIS
FUNCTIONAL SIM, INTERACTIVE TOOLS
OPERATING SYSTEM
FOR FLIGHT COMPUTERS, NETWORKS
INTEGRATED SUPPORT SOFTWARE SYSTEM
DATA BASE
SOFTWARE, DATA, VEHICLE, AND MGMT
MANAGEMENT SYSTEM
CONFIG. CONTROL, SCHEDULING, AND STATUS
CREW TRAINING
"COCKPIT", TRAINER CONTROLS
DATA REDUCTION
"POST-FLIGHT", TELEMETRY, SIMS PLOTS & PRINT
OPS PLAN
FLIGHT PLANNING, CREW ACTIVITIES AND PAYLOADS

ENVIRONMENT COMPONENTS

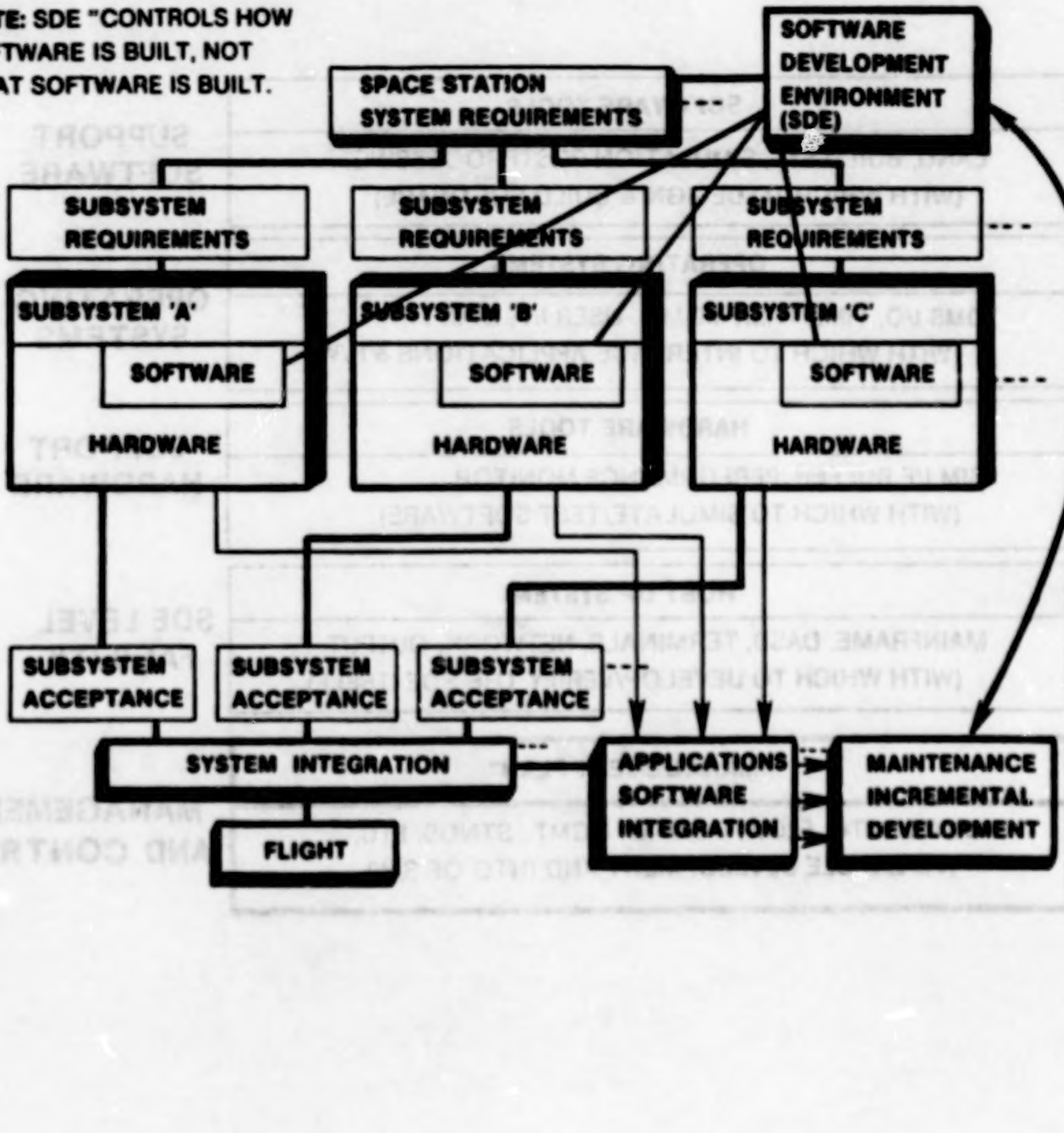
The software support environment will consist of five major constituents. They are software tools, operating systems, various hardware tools (such as simulation interface buffers and performance monitors), a host data processing system, and then last, and certainly not least, overall management policies, procedures, and standards. The management of the software development and acquisition process is critically important. Thus this last category, the "management plan" box, is highlighted.


SOFTWARE TOOLS
LANG, BUILD SYS, SIMULATION POSTPROCESSING. . .
(WITH WHICH TO DESIGN & BUILD SOFTWARE)
SUPPORT SOFTWARE
OPERATING SYSTEMS
DMS I/O, TIME/TASK MGMT., USER I/F, ETC. . .
(WITH WHICH TO INTERFACE APPLICATIONS & H/W)
OPERATING SYSTEMS
HARDWARE TOOLS
SIM I/F BUFFER, PERFORMANCE MONITOR. . .
(WITH WHICH TO SIMULATE/TEST SOFTWARE)
SUPPORT HARDWARE
HOST DP SYSTEM
MAINFRAME, DASD, TERMINALS, NETWORK, OUTPUT. . .
(WITH WHICH TO DEVELOP/VERIFY THE SDE ITSELF)
SDE LEVEL FACILITY
"MANAGEMENT PLAN"
DEVEL/INTG. FLOW, CONFIG. MGMT., STNDS, ETC. . .
(TO ENABLE DEVELOPMENT AND INTG OF S/W)
MANAGEMENT AND CONTROL

RELATIONSHIP

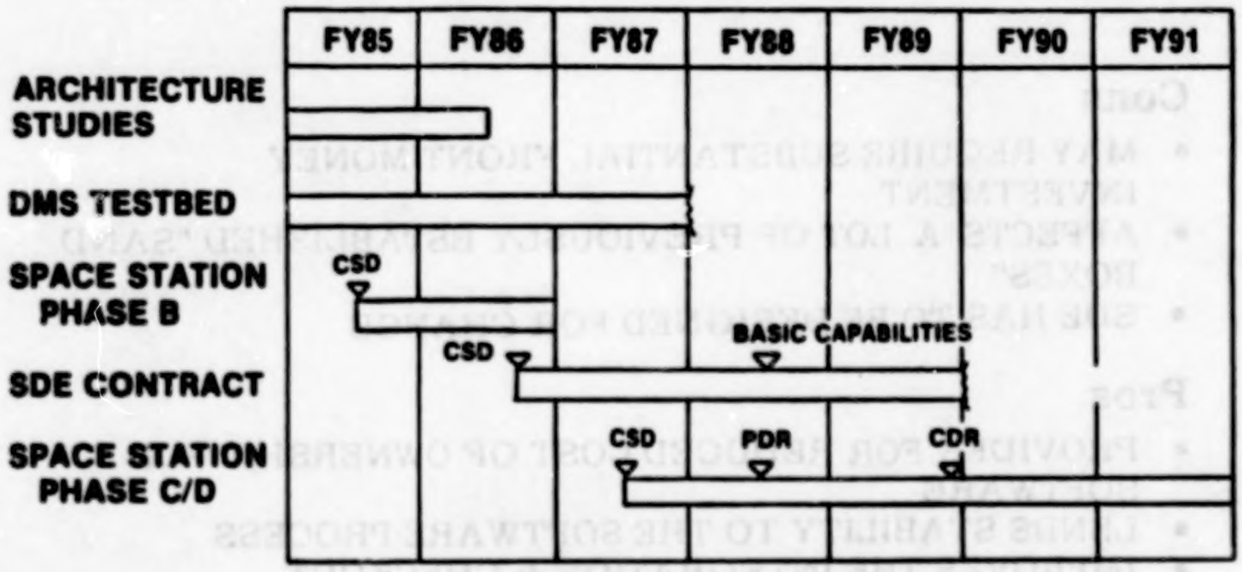
This figure tries to conceptualize how the software development environment will provide support throughout the system life cycle. First, the SDE will support each subsystem as it is being developed. That same software development environment will then provide support as those subsystems are integrated to form the system and then later on as that system moves into the long term maintenance and enhancement phase. A key driver behind the SDE concept is to minimize maintenance costs.

NOTE: SDE "CONTROLS HOW SOFTWARE IS BUILT, NOT WHAT SOFTWARE IS BUILT.



SCHEDULE

This figure illustrates the planned schedule for the software development environment. A separate contract for the software development environment will be issued in the latter part of FY 1986 so that we have a basic capability SDE in place by mid-year FY 1988. Note how that correlates with the Space Station Phase C/D mainstream development. Phase C/D is scheduled to start at mid-year FY 1987 so it is important that the SDE be in place, tested and checked out shortly thereafter, i.e., prior to the critical design review for the Space Station. We are on a very tight schedule.



PROS AND CONS

The software development environment has a number of advantages as well as a few disadvantages. Some of the disadvantages are that it will require a large investment up front. Certainly the establishment of a standard set of tools, practices, policies, and techniques will affect a number of previously established "sand boxes," by which I mean the ways people have traditionally been doing business both within NASA as well as within industry. In addition, the SDE must be designed for changes; it won't be a fixed set of tools. On the advantages side, however, we are firmly convinced that the SDE will substantially reduce the cost of ownership for our software, the ownership (maintenance) cost that we're worrying about being something like 70 to 80 percent of the total life cycle outlay. The SDE will also lend stability to our software process by helping to assure that all participants are using the same set of tools, standards and techniques and it will thus improve the integration and checkout process. We believe the advantages, particularly in the long run, far outweigh the disadvantages.

Cons

- MAY REQUIRE SUBSTANTIAL FRONT-MONEY INVESTMENT
- AFFECTS A LOT OF PREVIOUSLY ESTABLISHED "SAND BOXES"
- SDE HAS TO BE DESIGNED FOR CHANGE

Pros

- PROVIDES FOR REDUCED COST OF OWNERSHIP FOR SOFTWARE
- LENDS STABILITY TO THE SOFTWARE PROCESS
- IMPROVES THE INTEGRATION & CHECKOUT PROCESSES

Fact of Life:

THE CONTINUING RAPID EVOLUTION OF THE COMPUTING INDUSTRY

ISSUES

There are a number of issues associated with the software development environment. The first concerns the practicality of an SDE and whether NASA really should try to define and develop such a software support capability. Secondly, should we try to apply that software development environment to all software, both in-house and that which we contract for? What will be the impact of a NASA defined SDE on our contractor colleagues? Should NASA furnish the SDE, lock, stock and barrel, or only specify what it should be and allow each organization that wants a copy to procure their own software/hardware? Should the SDE be a single centralized facility or should we allow multiple copies of the SDE? Another very important question is how do we maintain configuration control? The SDE certainly won't be a static capability. What will be the government's liability? When software is late or has problems, will the developer be inclined to point to the SDE as a source of the problem? And then finally, a remaining issue is whether we really should be talking about two different kinds of SDE's, one that would support software development and the other that will support software acquisition and thus be largely a management SDE.

- **Should a uniform NASA SDE be defined and developed?**
- **Apply to all software development (in-house and contractors)?**
- **Relationship to established industry SDEs?**
- **NASA GFE or only specs?**
- **One central facility or multiple copies?**
- **How to configuration control?**
- **Government liability?**
- **Two SDEs: development and management?**

STANDARDS

The basic question concerning standards is what standards are needed. I have listed on this figure a few of the types of standards that we think we should have. This list ranges from types of documentation and formats for those documents down to terminology instruction, set architectures, standardized languages, standards for quality assurance, testing procedures, and a standardized life cycle. Now, in a couple of cases, we have already moved forward to begin the standardization process. We have established a standard life cycle, as shown on a previous figure. We are specifying a critical set of documents that should be required of most software projects. (It will always be possible to apply for a waiver, but we do have a standard set of documents that will normally be required.) We are also in the process of finalizing a software terminology standard. But what other categories should we be worrying about and what candidates exist to fill those needs?

- **WHAT STANDARDS ARE NEEDED?**
  - **Documentation types and formats?**
  - **Terminology?**
  - **16 bit and 32 bit instruction set architectures?**
  - **Languages?**
  - **Operating systems? Tools? DBMS?**
  - **Quality assurance?**
  - **Configuration management?**
  - **Testing procedures?**
  - **Life cycle (phases, events, products)?**
- **WHAT OTHERS?**
- **WHAT CANDIDATES FILL THE NEED?**

ARGUMENTS FOR AND AGAINST

There are a number of arguments leaning in favor of standards and of course some arguments against standards. Arguments that indicate that we should have standards point out that we will have greater compatibility in our equipment and data. It will be less costly to transfer information if we have standardized software/hardware and standardized documents. Systems and subsystems should be implemented more quickly. Standards should facilitate wider use of information, particularly across the large number of organizations that will comprise the Space Station Program. Standards in some areas at least will mean that we will need fewer skilled personnel. In other words, we won't have to train and maintain so many specialists in so many different areas. Arguments against standardization include the possibility of discouraging individual preference, moving us away from the leading edge of technology, and lowering the competitiveness of hardware and software.

The Argument for:

- **COMPATIBILITY FOR EQUIPMENT AND DATA**
- **LESS COSTLY TO TRANSFER INFORMATION**
  - NO NEED TO PURCHASE S/W, H/W BRIDGES
  - LESS PROGRAMMER TIME REQUIRED
- **FASTER IMPLEMENTATION**
- **WIDER USAGE OF INFORMATION**
- **LESS SKILLED PERSONNEL REQUIRED**

The Argument Against:

- **DISCOURAGES INDIVIDUAL PREFERENCE**
- **MOVES AWAY FROM "LEADING EDGE" OF TECHNOLOGY**
- **LOWERS COMPETITIVENESS OF HARDWARE AND SOFTWARE**

LANGUAGES

It has been the intent of the Space Station Program for some time now to standardize on a very few computer languages: one or two languages in the implementation category and a similar small set of languages in each of the other categories. But there are some basic questions that we must ask ourselves. One is should the Space Station Program try to standardize on languages at all? And if you agree that we should, then by what criteria? How is it that we should select one language versus another? And in each category of application, should we focus on one single language or a small set? Some considerations to fold in to our thinking about those questions include the fact that we want to minimize life cycle cost. This is a program that will stretch out over 25 or 30 years. The languages that we pick must be easy to use, and must be robust and have a wide range of functional capabilities. Of course, we would like a language that's reasonably mature and therefore has a good tool support and experience base. The languages must be compatible with the types of computers that we will use, the environments within which that hardware will be exercised, and the existing software. The latter is a very important point for Space Station because we must interface with a number of software applications that are already existing and are written in a number of different languages. Another consideration is programmer availability.

- **QUESTIONS**
  - **Should Space Station Program standardize languages at all?**
  - **If so, by what criteria?**
  - **One language or several?**
- **CONSIDERATIONS**
  - **Minimize life cycle costs**
  - **Ease of use**
  - **Richness and functional capabilities**
  - **Maturity and support base**
  - **Compatibility to machines, environments, other languages**
  - **Programmer availability**

CATEGORIES

Listed on this figure are the probable major categories for language standardization and a few of the possible candidates that might be suitable for each category. Now, that list of candiates is by no means complete but at least some of the major ones are listed. The categories are requirements and specifications, design, development (which is of course the language standardization area that people most often think of), the user interface, and artificial intelligence and expert systems.

LANGUAGES

CATEGORIES	CANDIDATES?
Requirements and specification	PSL/PSA, SREM, SADT, CADSAT
Design	PDL, SDDL
Development	HAL/S, Fortran, PL/I, Jovial, Ada, C, Modula-2, Pascal
User interface	GOAL, ATLAS, SCOL,STOL, Ada
AI/expert systems	LISP, PROLOG

ISSUES

There are a number of issues associated with selecting computer languages. The first one that comes to everyone's mind is Ada. Is Ada sufficiently mature? Does it have the proper set of tools available? If we decide not to follow the Ada route, at least for a period of time, then what languages or language should we be choosing temporarily? Another issue is how do we maintain language configuration control? How do we prevent or should we even try to prevent people from creating special versions of the selected standard language or languages? Other important issues revolve around the special application areas of expert systems, artificial intelligence, and the user interface. Do we need to select special languages for those categories or can the same standard language that we choose for implementation also suffice?

- Ada: Maturity

Tool set

If not Ada, what?

- How to maintain language configuration control?

- Languages for special purposes:

e.g., Expert systems

User interface

CONCLUSIONS

I have tried in this briefing to prompt your thinking. I have pointed out that software will be a very critical element of Space Station, prevalent throughout all aspects in space as well as on the ground. There are many open issues that the Program is now identifying and attempting to resolve. They range across the four major categories that will be the focus for this forum: software management planning, the software development environment, standards, and languages. We are requesting industry and university assistance and welcome your contributions.

- **SOFTWARE CRITICAL ELEMENT OF SPACE STATION**
- **MANY OPEN ISSUES**
  - **Management planning**
  - **Software development environment**
  - **Standards**
  - **Languages**
- **NASA REQUESTING INDUSTRY AND UNIVERSITY OPINION AND IDEAS**

SOFTWARE MANAGEMENT PANEL SUMMARY

Prior to the forum, the Software Management Panel reviewed the Space Station Software Issues report (ref. 1) and the draft Level A/B Software Management Plan (Table 1). During the forum, the panel experts and the audience made 30 specific recommendations for assuring the successful management of Space Station software. The following six recommendations are essential to the Program's success and are the basis for accomplishing the 30 specific recommendations.

1. The charters of the Level A and B Software Managers must be strengthened to assure that those positions have the decision and control authority to properly conduct their jobs. Specific actions are:

   a. Support the Level A and B Software Managers with increased software-experienced staff. (The panel notes with alarm the lack of any support staff at the present time for the Level A position.)

   b. The Level A and B Software Managers must each have significant discretionary budget to provide the appropriate guidance and support of the software management and acquisition functions below them.

   c. The hierarchy of software decision making and approval authority must be clearly established. The panel recommends that technical decisions with system engineering and integration impact be the responsibility of the Level B Software Manager with the concurrent involvement of the Level A Software Manager. However, the panel recognizes that there will be certain major decisions (such as the choice of a standard language and the overall concept for the software support environment) that will have major, long reaching impact, both within the Program and to organizations that interface to the Program. The panel recommends that such decisions be the responsibility of the Level A Software Manager with the concurrent involvement of the Level B Software Manager.

   d. The Software Management Plan needs to be modified as follows:

      - Develop and adopt charter statements for both the Level A and B Software Managers.
      - Specify items a, b, and c above in the charter statements.
      - Identify and provide a schedule for important decisions that need to be made.
      - Specify how the decisions will be made and by whom.
      - Specify who has control of the management functions, e.g., budget approval and product approvals.

2. The Software Management Plan policies and procedures are in-house development oriented, whereas in fact the task is the management of the acquisition of software (including in-house development). Large-scale software acquisition is new to some parts of NASA and is different from, and more difficult than, hardware acquisition. The plan must be reformulated to reflect this acquisition orientation. Various sections in the plan need to be revised to strengthen the policies and the ability of the Level A and B Software Managers to be effective in playing a role in software acquisition. The plan should call for in-house (NASA) software development to be managed in the same way as non-NASA

(contractor) acquisition/development, with appropriate tailoring to accommodate differences such as legal contracting procedures for external acquisitions.

3. The focus of the Software Management Plan needs to be revised to emphasize the maintenance/sustaining engineering considerations in more detail and earlier in the system life cycle process. The major cost of most long-life-cycle computer-based systems is in the post-delivery-to-operations phase (60-80% of total software life cycle costs). The role of the software managers in the early system definition and design phase should be expanded to provide for software allocation and software trade-offs. If the wrong decisions are made in this phase, it will be nearly impossible to reduce the maintenance/sustaining engineering costs later.

4. It is not clear what the boundaries of Space Station are. The specific management spheres of control are unclear and the procedures for accomplishing management interaction with non-Space Station services are not defined. Additionally, much of the inherited software appears to be outside the control of Space Station policies and standards. For example, interoperability design, interface design, and integrated schedule coordination need to be more clearly delineated. Policies and procedures for managing these issues must be specified as they impact Space Station software.

5. The Software Management Plan and stated NASA approach call for NASA to perform the top level software engineering and integration (SE&I) function. The panel observes that the scope of that task (multicenter, multicontractor and multi-subcontrator) is far beyond NASA's past experience. The panel suggests that the full scope of the SE&I job be re-assessed with special attention to integration. The plan must address more specifically the management of the many geographically dispersed organizations involved in the integration task. More detail is needed on policies (who, how, when) and on the specific responsibilities of developers and integration organizations.

6. It is the consensus of the panel that the Software Management Plan should be restructured. A new table of contents is recommended that provides for:

   - A more complete list of policies.
   - Charters for the Level A and B Software Managers that are sufficient and delimiting with respect to control and authority of the management process.
   - Special attention and focus on several significant procedures.

RECOMMENDATIONS

1. The Level A/B Software Management Plan structure does not focus sufficient emphasis on several areas and needs revision. (See Table 1, recommended Software Management Plan Table of Contents, p. 76.)

2. The interdisciplinary activities and interactions are not well defined. Their definition and control mechanisms should be specifically detailed in the Level A/B Software Management Plan.

3. The Level A/B Software Management Plan should emphasize the considerations of using existing (inherited) software as an alternative to totally new development.

4. The Level A/B Software Management Plan should specify the policies and procedures for control and feedback between the level A/B/C management functions for cost, schedule and technical content.

5. The Level A/B Software Management Plan should specify the policies and procedures for managing the risk issues.

6. The Level A/B Software Management Plan should specify the policies and procedures for managing the various technical performance items.

7. The Level A/B Software Management Plan should address the policies and procedures to accommodate modern, appropriate software development methodologies.

8. The Level A/B Software Management Plan should focus more emphasis on the early planning for the maintainability/sustainability aspects of acquired software.

9. The policies on independent verification and validation (IV & V) in the Level A/B Software Management do not put enough emphasis on its SELECTIVE use. The criteria for utilization of IV&V should be defined.

10. The policies and procedures for managing the acquisition and configuration management of FIRMWARE should be specified.

11. The Level A/B Software Management Plan policies and procedures for acquisition of software should emphasize QUALITY and should be formulated and reviewed to accommodate new paradigms as they may be accepted industry practice over the life of the project (30+ years).

12. The policies and procedures in the Level A/B Software Management Plan should specify how and when software and hardware trade-offs are made in the system life cycle, as well as how and when hardware/software interfaces are defined.

13. The Level A/B Software Management Plan policies and procedures for tailoring should set tailoring guidance based upon different identified categories of software and should provide different life cycles if appropriate.

14. The Level A/B Software Management Plan should define the policies and procedures for the various reviews addressing the who, why, what, and when. They should also provide for an evaluation of the review process and a mechanism for improving the review process.

15. The Level A/B Software Management Plan should specify the policies and procedures for contract incentives that are easily understood and administered and are directly tied to the cost, schedule and technical content, and quality of the product.

16. The Level A/B Software Management Plan needs to stress the policies and procedures for ACQUISITION of software rather than DEVELOPMENT of software.

17. The Level A/B Software Management Plan should rely heavily on existing government and industry standards such as the new DOD-STD 2167 (ref. 2) and IEEE standards.

18. The life cycle definition should expand its scope to include the system front-end definition and design, operations, and sustaining engineering, and to specify the products and reviews relevant to each phase.

19. The Level A/B Software Management Plan should specify the policies and procedures for defining and managing the support system interfaces and interoperability, such as TDRSS, Shuttle, Mission Control, etc.

20. The Level A/B Software Management Plan should first focus on the acquisition/development methods and languages and then choose the tools to support the methods for the Software Management Environment.

21. The Level A/B Software Management Plan should specify the procedures for its timely review, approval, and maintenance.

22. The Level A/B Software Management Plan should specify policies and procedures based on legal and government policies for managing the software on an international basis with respect to proprietary information and software and the export of key US technology.

23. The Level A/B Software Management Plan should address the policies and procedures for managing the security, sensitivity, privacy, and contamination/destruction issues of software acquisition and ownership.

24. The Level A/B Software Management Plan should specifiy the policies and procedures for the decision process and authority for decision making.

25. The Level A/B Software Management Plan should specify the policies and procedures for insuring non-loss of software and continuous operations due to inadvertent and/or catastropic loss of operational or support software.

26. The Level A/B Software Management Plan policies and procedures should focus on the management, control, quality, etc. of the PRODUCTS as opposed to the development process; i.e., acquisition management as opposed to development management.

27. The Level A/B Software Management Plan should specify the policies and procedures for "designing-to-cost" as a potential acquisition strategy.

28. The Level A/B Software Management Plan should specify the primary goals and objectives of the plans, policies, and procedures.

29. The Level A/B Software Management Plan should specify the policies and procedures for obtaining and utilizing software acquisition experience from past and future projects.

30. The Level A/B Software Management Plan should specify the policies and procedures for establishing standardization.

D2

N86-23316

A VIEW OF SOFTWARE MANAGEMENT ISSUES

John H. Manley
Computing Technology Transition, Inc.,
and Nastec Corp.

FOREWORD

The following briefing charts have been supplemented with post-forum comments to both emphasize and clarify some of the key points.

PRESENTATION TOPICS

- O MANAGEMENT BRIEFING AND PANEL OBJECTIVES
- O LARGE SOFTWARE SYSTEM MANAGEMENT ISSUES
- O NASA-DEFINED MANAGEMENT ISSUES AND SOLUTIONS
- O INITIAL REACTION TO NASA PROPOSALS
- O ADDITIONAL SOFTWARE MANAGEMENT ISSUES
- O SUMMARY VIEWS OF NASA SOFTWARE MANAGEMENT ISSUES
- O INITIAL RECOMMENDATIONS

The presentation topics shown here are intended to provide a sequence of discussion which sets the stage for the subsequent open and closed panel sessions on software management issues. The purpose of these sessions is to provide an objective industry-oriented critique of NASA-defined management issues contained in both reference 1 and the "Preliminary Space Station Level A/B Software Management Plan."

MANAGEMENT BRIEFING OBJECTIVES

- O SUMMARY ASSESSMENT OF "SPACE STATION SOFTWARE ISSUES" REPORT
- O CRITIQUE OF ISSUES AND PROPOSED SOLUTIONS
- O ADDITIONAL SIGNIFICANT ISSUES THAT NASA SHOULD CONSIDER
- O RELEVANCE OF ISSUES TO CURRENT R&D IN INDUSTRY AND ACADEMIA
- O OPENING BRIEFING AND NASA REPORT FORM BASIS FOR DISCUSSION IN FIRST CLOSED PANEL SESSION

The objectives shown here are intended to provide a basis for initial management panel discussions. During that discussion, the other panel members will add to or revise the issues contained in this briefing in order to present a comprehensive set of issues to the open session attendees for their response.

MANAGEMENT PANEL OBJECTIVES

- O SUMMARIZE AND SUPPLEMENT NASA-DEFINED MANAGEMENT ISSUES
- O PROVIDE INDUSTRY REACTION TO PLANNED POLICIES AND APPROACH
  - REASONABLE?
  - LIKELY TO WORK?
  - ACHIEVE GOAL OF MINIMIZING SOFTWARE OWNERSHIP COST?
- O CRITIQUE PLAN OF SOFTWARE DEVELOPMENT AND MANAGEMENT STRATEGY
  - STRENGTHS?
  - WEAKNESSES?
  - DISAGREEMENTS?
- O RELEVANCE OF ISSUES TO CURRENT R&D EFFORTS
  - INDUSTRY?
  - ACADEMIA?
  - GOVERNMENT?

The industry reaction to NASA plans is extremely important in helping to identify the relevance of their proposed activities to similar steps being taken elsewhere, e.g., industry organizations such as the MCC in Austin, Texas, and the newly proposed Software Productivity Consortium, as well as the Department of Defense software initiatives of Ada, STARS and the Software Engineering Institute. Since NASA has international partners, the U.K.'s Alvey program, the EEC's ESPRIT program, and the Japanese fifth generation computer project also have relevance to Space Station software technology. This is particularly important with regard to the management of new technology transition, or insertion, into Space Station during its formative years.

LARGE SOFTWARE SYSTEM MANAGEMENT ISSUES

SPECIAL CHALLENGES

- MUST SOLVE COMPLEX PROBLEMS
- REQUIRES COOPERATIVE LABOR
- SOLUTIONS OFTEN COUNTERINTUITIVE
- RIGID DEVELOPMENT AND SUPPORT PROCESSES
- EXPENSIVE PRODUCTION AND SUPPORT
- HIGH RISK

HENCE

LARGE SYSTEMS ARE

VERY DIFFICULT TO MANAGE

Space Station is an extremely complex and large undertaking. It will contain subsystems containing large to super-large software components that must be integrated in a logical manner. Since the total architectual design is beyond any single human's comprehension, these typical large system problems will be encountered by NASA management. The job will be very difficult and should be recognized at the outset.

LARGE SOFTWARE SYSTEM MANAGEMENT ISSUES

(CONTINUED)

TYPICAL MANAGEMENT PROBLEMS ON VERY LARGE PROJECTS

- CONTINUING REQUIREMENTS CHANGES
- UNEXPECTED GROWTH IN CODE SIZE
- DOCUMENTATION OVERLOADS
- HIGH TRAVEL COSTS (BOTH DOLLARS AND TIME)
- INTEGRATION AND TEST OVERLOADS
- UNEXPECTEDLY HIGH ERROR RATES
- POOR HUMAN FACTORS
- SCHEDULES OUTSIDE OF PROJECT CONTROL
- DELIVERY MUCH LATER THAN REQUIRED
- UNSUPPORTED, UNTRAINED SUSTAINING ENGINEERING PERSONNEL
- LOW MORALE AND HIGH TURNOVER

NASA management can expect to encounter most if not all of the problems shown on this list. By anticipating such problems, NASA will be better equipped to satisfactorily identify their early symptoms, deal with them in an orderly way (perhaps through the exercise of contingency plans), and prevent any software crisis from disrupting the program.

LARGE SOFTWARE SYSTEM MANAGEMENT ISSUES

(CONTINUED)

IMPORTANT CONSIDERATIONS

- O PRODUCT MANAGER(S)
  - RESPONSIBILITY
  - AUTHORITY
  - EXPLICIT DELIVERABLES
- O TOP MANAGEMENT COMMITMENT TO PROCESS
  - IMPLEMENT
  - USE
  - ENFORCE
- O PRODUCT MANAGEMENT PROCESS INTEGRATION
  - HARDWARE
  - SOFTWARE
  - SYSTEMS
- O FLEXIBILITY IN STANDARDS APPLICATION
  - LARGE VERSUS SMALL PROJECTS
  - NEW VERSUS ENHANCED PROJECTS
  - MULTI-SITE, MULTI-CONTRACTOR DEVELOPMENT
  - DIFFERENT PRODUCT TYPES
    - -- SOFTWARE ONLY
    - -- HARDWARE/SOFTWARE

The most important of the "important considerations" shown here is the product orientation. By product I mean platforms, modules, maneuvering vehicles, and so forth that are dependent upon highly reliable, fault tolerant, adaptable software systems. Furthermore, since Space Station is composed of a collection of fully integrated hardware/software/human systems, NASA cannot artificially separate software from such systems except where it makes sense.

NASA DEFINED MANAGEMENT ISSUES

O SOFTWARE MANAGEMENT PLANNING

- SOFTWARE MANAGEMENT PLAN
- IMPLEMENTATION BY NASA AND CONTRACTORS
- UPPER MANAGEMENT EDUCATION
- TRAINING AT ALL LEVELS

O INDEPENDENT VERIFICATION AND VALIDATION

- WHERE SHOULD IV&V BE USED?
- HOW SHOULD IT BE MECHANIZED?
- RELATIONSHIP TO SOFTWARE DEVELOPMENT ENVIRONMENT

O QUALITY ASSURANCE AND CONFIGURATION MANAGEMENT

- ROLE OF QUALITY ASSURANCE ORGANIZATIONS
- TRAINING AND PREPARATION
- LEVEL OF REQUIRED CONFIGURATION CONTROL
- DEGREE OF NASA INVOLVEMENT

O AVOIDING MAJOR SOFTWARE PROBLEMS

- RISK AVOIDANCE
- RISK CONTAINMENT

The issues defined here are what I considered the major topics contained in the NASA planning documents. Many other issues were defined as well.

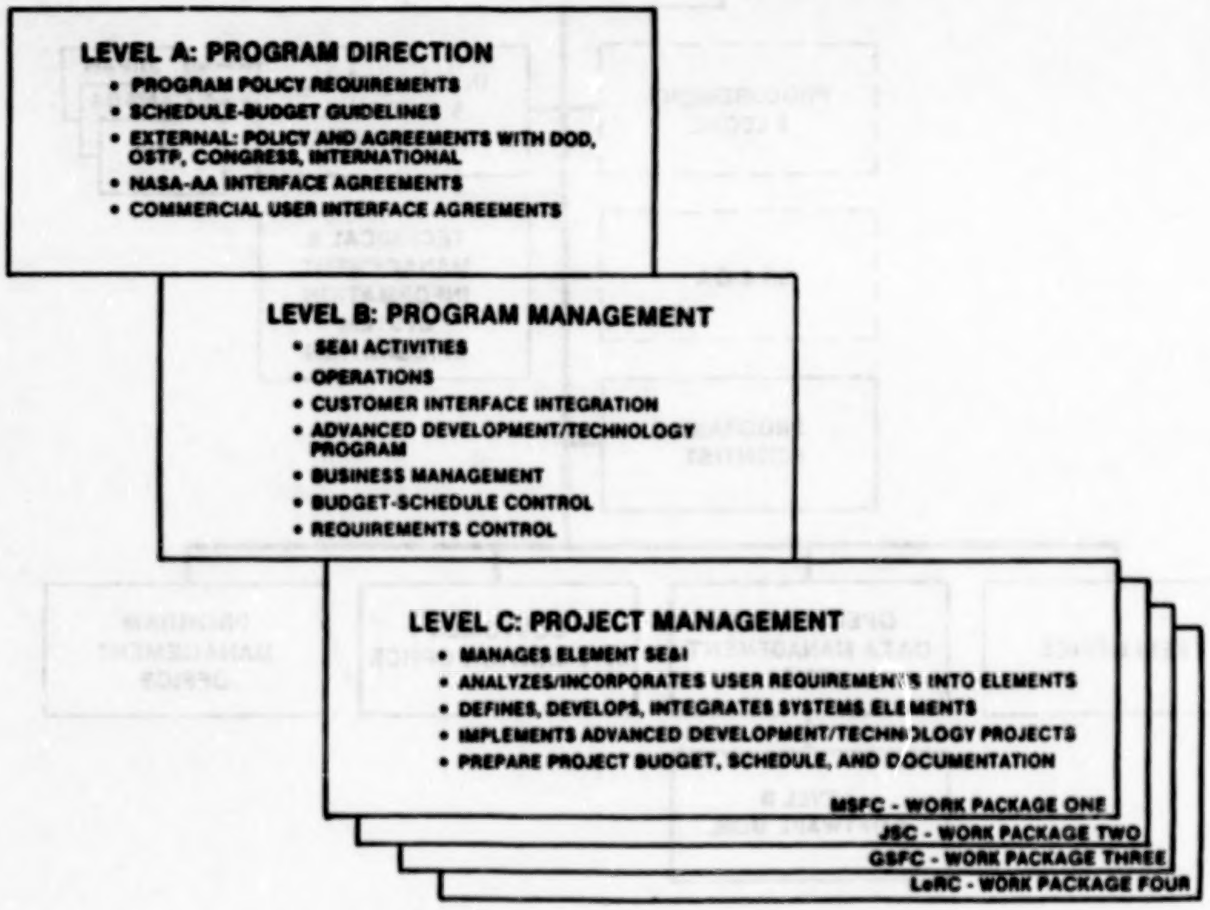
NASA PROPOSED SOLUTIONS

- O THREE-LEVEL MANAGEMENT STRUCTURE WITH ELABORATE PLANNING SYSTEM
- O NASA SOFTWARE LIFE CYCLE FRAMEWORK
- O HEAVY EMPHASIS ON INDEPENDENT VERIFICATION AND VALIDATION OF SOFTWARE (IV&V)
- O STRINGENT CONFIGURATION CONTROL SYSTEM
- O NASA-SPONSORED MANAGEMENT TOOLS AND PRACTICES DATABASE

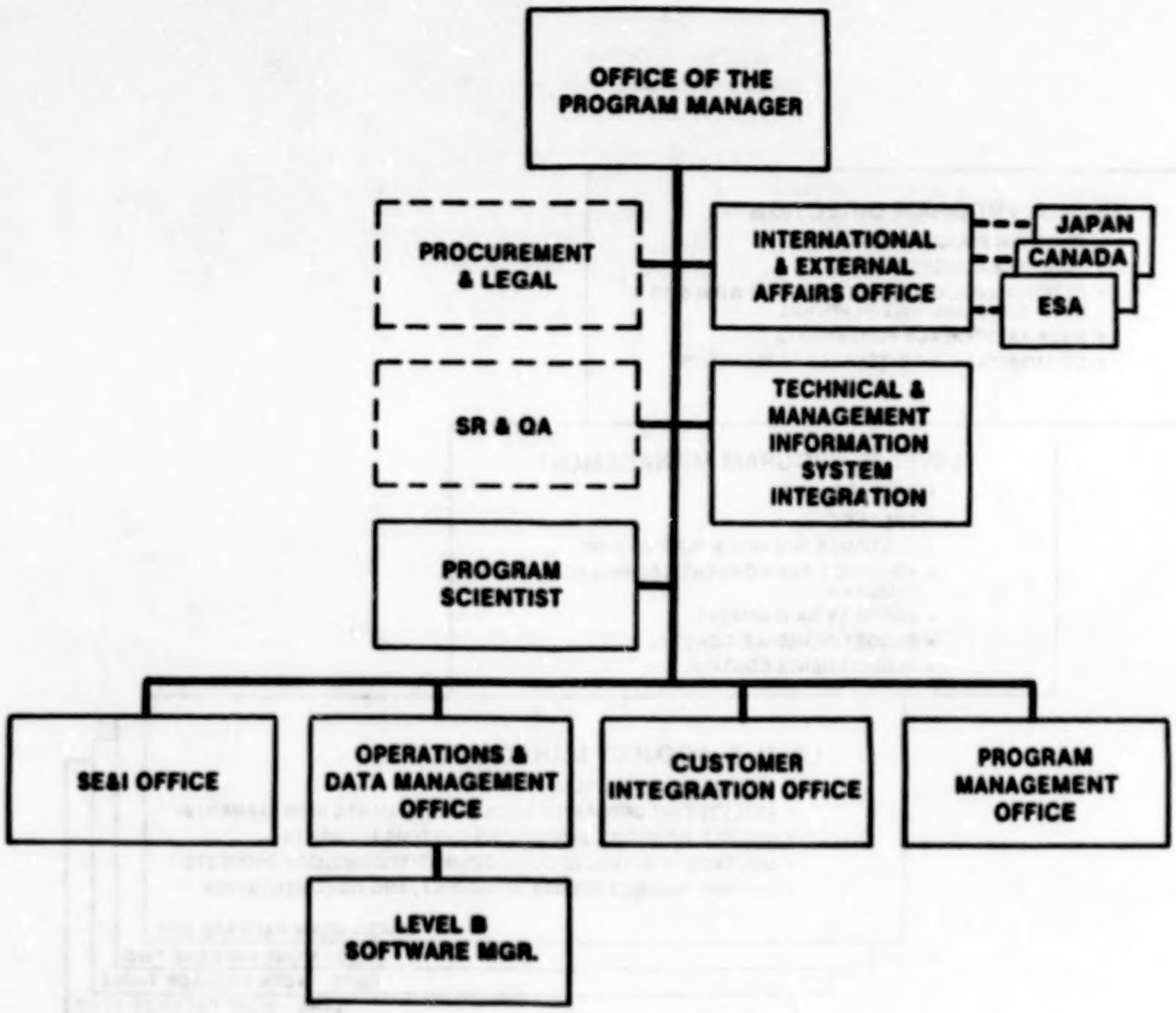
These are the key proposals contained in the draft management plan.

The next five figures have been extracted from the NASA draft management plan and illustrate the detailed thinking that has gone into the planning process.

This figure and the one on the following page show a three-level management structure, from policy making to software acquisition management. A question arises with respect to how clear lines of authority and responsibility will be implemented within the very complex office structures proposed for the program. What is line and what is staff? Who has authority in addition to responsibility?


LEVEL A: PROGRAM DIRECTION
• PROGRAM POLICY REQUIREMENTS
• SCHEDULE-BUDGET GUIDELINES
• EXTERNAL: POLICY AND AGREEMENTS WITH DOD, OSTP, CONGRESS, INTERNATIONAL
• NASA-AA INTERFACE AGREEMENTS
• COMMERCIAL USER INTERFACE AGREEMENTS
LEVEL B: PROGRAM MANAGEMENT
• SE&I ACTIVITIES
• OPERATIONS
• CUSTOMER INTERFACE INTEGRATION
• ADVANCED DEVELOPMENT/TECHNOLOGY PROGRAM
• BUSINESS MANAGEMENT
• BUDGET-SCHEDULE CONTROL
• REQUIREMENTS CONTROL
LEVEL C: PROJECT MANAGEMENT
• MANAGES ELEMENT SE&I
• ANALYZES/INCORPORATES USER REQUIREMENTS INTO ELEMENTS
• DEFINES, DEVELOPS, INTEGRATES SYSTEMS ELEMENTS
• IMPLEMENTS ADVANCED DEVELOPMENT/TECHNOLOGY PROJECTS
• PREPARE PROJECT BUDGET, SCHEDULE, AND DOCUMENTATION
MSFC - WORK PACKAGE ONE
JSC - WORK PACKAGE TWO
GSFC - WORK PACKAGE THREE
LeRC - WORK PACKAGE FOUR

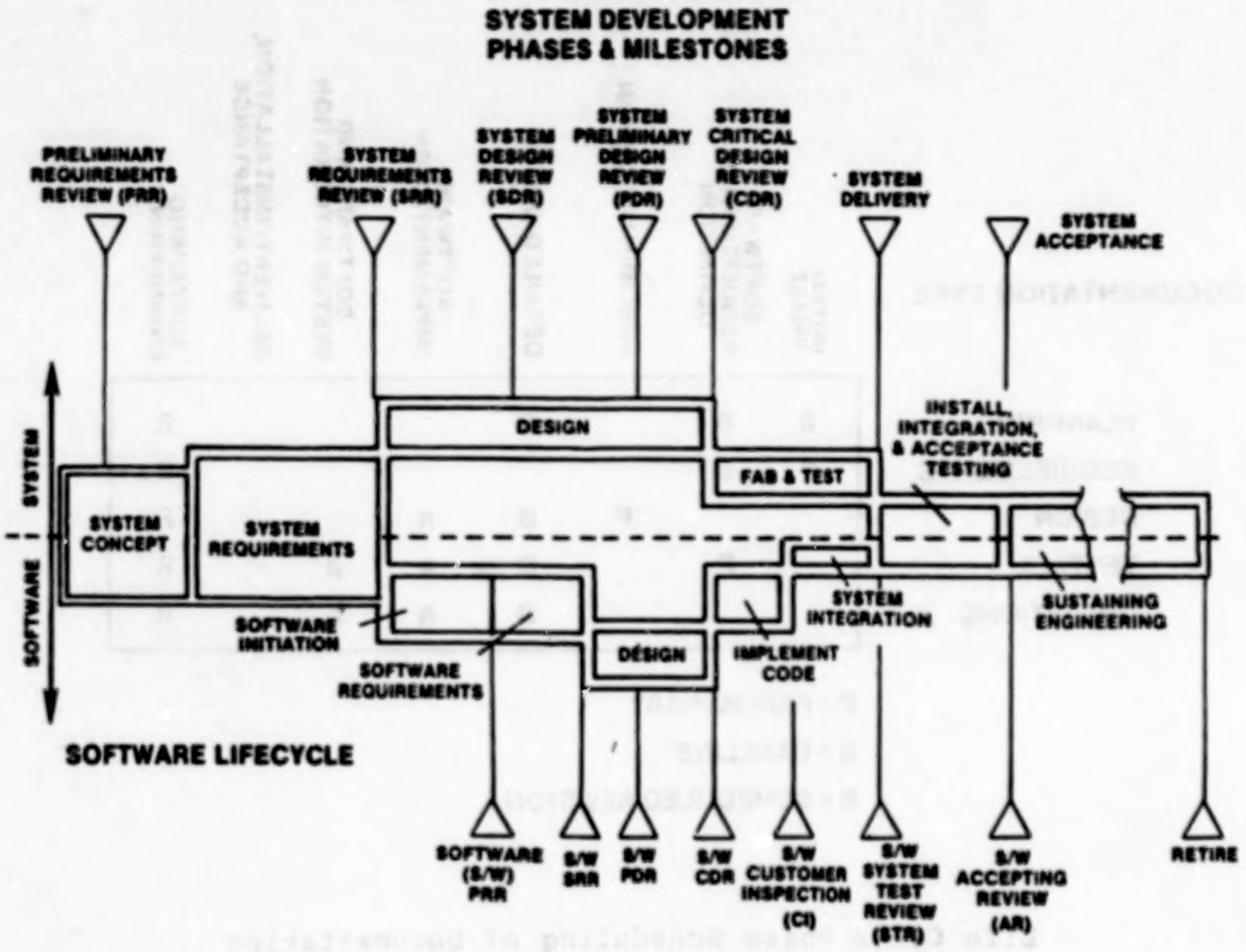
Space Station Program organization structure and hierarchy



OFFICE OF THE PROGRAM MANAGER
PROCUREMENT & LEGAL
INTERNATIONAL & EXTERNAL AFFAIRS OFFICE
JAPAN
CANADA
ESA
SR & QA
TECHNICAL & MANAGEMENT INFORMATION SYSTEM INTEGRATION
PROGRAM SCIENTIST
SE&I OFFICE
OPERATIONS & DATA MANAGEMENT OFFICE
CUSTOMER INTEGRATION OFFICE
PROGRAM MANAGEMENT OFFICE
LEVEL B SOFTWARE MGR.

Level B Space Station Program Office structure

The life cycle is very important to NASA for many reasons. However, I question the starting point for software in the Design Phase. I recommend that software activities be included as early as the Preliminary Requirements Review phase.


SYSTEM DEVELOPMENT PHASES & MILESTONES
PRELIMINARY REQUIREMENTS REVIEW (PRR)
SYSTEM REQUIREMENTS REVIEW (SRR)
SYSTEM DESIGN REVIEW (SDR)
SYSTEM PRELIMINARY DESIGN REVIEW (PDR)
SYSTEM CRITICAL DESIGN REVIEW (CDR)
SYSTEM DELIVERY
SYSTEM ACCEPTANCE
SYSTEM
SOFTWARE
DESIGN
INSTALL, INTEGRATION, & ACCEPTANCE TESTING
FAB & TEST
SYSTEM CONCEPT
SYSTEM REQUIREMENTS
SOFTWARE INITIATION
SOFTWARE REQUIREMENTS
DESIGN
IMPLEMENT CODE
SYSTEM INTEGRATION
SUSTAINING ENGINEERING
SOFTWARE LIFECYCLE
SOFTWARE (S/W) PRR
S/W SRR
S/W PDR
S/W CDR
S/W CUSTOMER INSPECTION (CI)
S/W SYSTEM TEST REVIEW (STR)
S/W ACCEPTING REVIEW (AR)
RETIRE

Space Station Program Software Life Cycle Phase

NASA has done a good job in identifying necessary documents, where they should be used, and how they should be controlled.

	SOFTWARE LIFE-CYCLE PHASE							
DOCUMENTATION TYPE	INITIAL PHASE	SOFTWARE REQUIREMENTS DEFINITION	PRELIMINARY DESIGN	DETAILED DESIGN	SOFTWARE IMPLEMENTATION	SOFTWARE AND SYSTEM INTEGRATION	DELIVERY, INSTALLATION, AND ACCEPTANCE	SUSTAINING ENGINEERING
PLANNING	B	R		R				R
REQUIREMENTS	P	B						R
DESIGN			P	B	R			R
TESTING		P		B	R	R		R
OPERATIONS				B	R	R		R

P = PRELIMINARY

B = BASELINE

R = SCHEDULED REVISION

Life Cycle Phase Scheduling of Documentation Requirements

	PRELIMINARY REQ'TS. REVIEW	SYSTEM REQ'TS. REVIEW	PRELIM. DES REVIEW	CRITICAL DESIGN REVIEW	CONFIGURATION INSPECTION	SYSTEM TEST REVIEW	ACCEPTANCE REVIEW	SUSTAINING ENGINEERING
PLANNING								
S/W MANAGEMENT PLAN	B		R					R
S/W DEVELOPMENT PLAN	B		R					R
CONFIG MGMT PLAN	B							R
SE & I PLAN	B							R
INTERFACE CTL PLAN	B							R
SRM & QA PLAN	B							R
V & V PLAN	B			R				R
I V & V PLAN	B			R				R
FACILITY PLAN	B	R						
ADP ACQUISITION PLAN	B		R					R
S/W STANDARDS	B							R
REQUIREMENTS								
S/W CONCEPT DOC	B							R
S/W REQUIREMENTS SPEC		B					D	R
ICD'S		B						R
DESIGN								
S/W DESIGN DOC			B	B	R		D	R
PROCUREMENT DOC			B	R				
SUSTAINING ENG. PLAN				B		R		R
CODE					B	R	D	R
TESTING								
S/W TEST PLAN		B	R					R
S/W TEST REQUIREMENTS		B		R				
TRACEABILITY DOC		B	R	R	R	R	D	R
OPERATIONS								
USER'S GUIDE				B	R		D	R
OPERATIONS MANUALS					B		D	R
VERSION DESCRIPTION DOC						B	D	R
PROGRAMMER'S HANDBOOK				B	R		D	R
S/W TEST PROCS				B	R		D	R
ACCEPTANCE TEST PROCS.				B	R			R
REPORTS								
SOFTWARE REVIEW REPORTS	X	X	X	X	X	X	X	X
S/W TEST REPORTS					X	X	X	X
SRM & QA REPORTS		X	X	X	X	X	X	
CR'S	X	X	X	X	X	X	X	X
LESSONS LEARNED	X	X	X	X	X	X	X	X
ACCEPT. TEST REPORTS						X	X	X

B: BASELINE **X: REPORT MILESTONE**
R: REVISED **D: DELIVERABLE**

Software Life Cycle Documentation Matrix

INITIAL REACTION TO NASA PROPOSAL

SOFTWARE MANAGEMENT PLANNING

- O NASA MANAGEMENT APPROACH EMPHASIZES PANELS, COMMITTEES AND AN ELABORATE SYSTEM OF PLANS
- O TOO MUCH FAITH IN PLANS (PEOPLE NOT PAPER GET THINGS DONE)
- O WHAT NEEDS TO BE ADDED:
  - ASSIGN RESPONSIBILITY FOR DELIVERABLES
  - MAKE PEOPLE ACCOUNTABLE FOR THEIR DELIVERABLES
  - INSTALL A SOFTWARE SCORING SYSTEM TO KEEP TRACK OF THEIR PROGRESS
  - ASSIGN RESPONSIBILITIES FOR TAKING POSITIVE CORRECTIVE ACTIONS
  - MANAGE THE RESPONSIBLE PEOPLE

My initial reaction to NASA's planning approach is that they have spent considerable time defining their problems. Furthermore, they have proposed to solve these problems through an elaborate system of plans to be implemented by a complex of offices, panels and committees. My visceral reaction to this approach is that there might be an overemphasis on "paper" and not enough on "people." By this I mean the list of items above under "what needs to be added."

Of most importance is identifying specific people to carry out Space Station software acquisition/development and support responsibilities and giving them the resources and necessary authority to carry out their jobs effectively.

In addition, these people must be managed to include the installation and use of an accounting system so that problems (and successes) can be quickly identified and corrective actions expeditiously initiated whenever and wherever needed.

The fundamental point is that, although the planning effort so far looks good on the surface due to the great attention to detail in organization and documentation, the ultimate key to success will lie in NASA's effective use of people.

INITIAL REACTION TO NASA PROPOSAL

(CONTINUED)

LIFE CYCLE FRAMEWORK

O PROPOSED NASA SOFTWARE LIFE CYCLE FRAMEWORK IS ESSENTIAL

- **FORCES CONSCIOUS DECISION MAKING**
- **INTEGRATES/INTERRLATES FUNCTIONS (SOFTWARE DEVELOPMENT, HARDWARE ENGINEERING, BUDGETING, SUPPORT, etc.)**
- **IMPROVES PREDICTABILITY**
- **HELPS QUANTIFY RISKS**
  - **-- SCHEDULES**
  - **-- DEPENDENCIES OR EXPOSURES**
  - **-- TECHNOLOGY NEEDS**
- **BETTER CONTROL OF EXTERNAL COMMITMENTS**

The NASA software life cycle framework as proposed in the draft management plan is excellent and essential due to the points outlined here.

INITIAL REACTION TO NASA PROPOSAL
(CONTINUED)

INDEPENDENT VERIFICATION AND VALIDATION

O NASA EMPHASIS ON IV&V GOOD BUT STARTS TOO LATE IN THE LIFE CYCLE

- CAN NOT TEST IN QUALITY
- MUST VERIFY DESIGN IDEAS EARLIER IN PROCESS
- SOFTWARE MANAGER MUST BE INVOLVED IN SYSTEM REQUIREMENTS ANALYSIS AND EARLY DESIGN DECISIONS

O QUESTIONS TO ANSWER DURING PRODUCT CONCEPTUAL PLANNING

- WHAT IS IT? WHO WILL USE IT? WHEN? WHY?
- PRODUCT STRATEGY

O QUESTIONS TO ANSWER DURING PRODUCT REQUIREMENTS DEFINITION

- WHAT MUST IT DO? HOW WILL IT BE DESIGNED?
- HOW WILL IT BE DEVELOPED? SERVICED?
- COST AND SCHEDULE ESTIMATES
- FINANCIAL AND WORK PLAN
- INITIAL HARDWARE/SOFTWARE ALLOCATION

With regard to NASA's heavy emphasis on independent verification and validation of software, I agree with the approach due to the special requirements for ultra-reliable spaceborne system software.

On the other hand, IV&V should be started much earlier than proposed to address the issues raised on this chart.

INITIAL REACTION TO NASA PROPOSAL

(CONTINUED)

<u>CONFIGURATION CONTROL</u>

O NASA EMPHASIS ON CONFIGURATION CONTROL CORRECT

O AREAS FOR IMPLEMENTATION (NASA AND ALL CONTRACTORS)

- SOFTWARE CHANGE CONTROL
- DOCUMENT CONTROL
- RELEASE CONTROL
- LIBRARY CONTROL

NASA cannot put too much emphasis on configuration control. However, they must ensure that such activities not be restricted to controlling code alone, but also to documents, releases as entities, and even the libraries themselves.

INITIAL REACTION TO NASA PROPOSAL

(CONTINUED)

<u>TOOLS AND PRACTICES DATABASE</u>

O NASA-SPONSORED SOFTWARE MANAGEMENT TOOLS AND PRACTICES DATABASE AND INFORMATION RETRIEVAL SYSTEM

- WHO WILL USE THIS OTHER THAN RESEARCHERS?
- HOW WILL THIS HELP MANAGERS?

O NICE IDEA BUT VERY LOW LEVERAGE ITEM IN GETTING THE JOB DONE

O CHANNEL ENERGIES TO SUPPORT THESE FUNCTIONS INSTEAD

- PHASE REVIEW DOCUMENTATION SUPPORT SYSTEM
- DISTRIBUTED FAULT ANALYSIS AND REPAIR
- DISTRIBUTED INTEGRATION SUPPORT
- DISTRIBUTED FIELD MAINTENANCE SUPPORT
- DEVELOPMENT TOOL DISTRIBUTION

O AND ... DEVELOPING THESE COMMUNICATION BUILDING BLOCKS

- TERMINAL ACCESS
- INFORMATION TRANSFER
- FILE TRANSFER
- DISTRIBUTED EXECUTION

The software management tools and practices database is primarily a research oriented effort that should be left to the research community to carry out (especially if requested by NASA). The talents required to perform this proposed effort are too valuable to use in building a product that has a high probability of not being used by its intended customers, i.e., real world program, project and software engineering managers.

I suggest that NASA channel the energies of its talented database technicians into the functions outlined on the chart, to include developing some of the very formidable communication technology components indicated. These real products are vitally needed to support the extremely important configuration control systems cited previously.

ADDITIONAL MANAGEMENT ISSUES

O SOFTWARE ACQUISITION POLICIES AND PRACTICES

- RIGHTS IN DATA
- SECURITY
- INCENTIVES
- SUBCONTRACTOR CONTROL
- ACCEPTANCE PROCESS
- WARRANTIES

O STANDARDIZATION

- LIFE CYCLE PROCESS
- CONTRACTING
- COST AND SCHEDULE REPORTING
- PROGRAM REVIEWS AND AUDITS

O GOVERNMENT FURNISHED MATERIALS

- SOFTWARE DEVELOPMENT
- SUSTAINING ENGINEERING

O PRODUCT CONTROL

- ARCHITECTURAL CONTROL
- VERSION CONTROL
- INTERFACE CONTROL

This is simply a partial but very important list of more issues that NASA Space Station software management must be concerned with. Each one was elaborated in the original briefing and in the panel discussions that followed.

SUMMARY VIEW OF SOFTWARE MANAGEMENT ISSUES

NASA'S PRIMARY CHALLENGE

SOFTWARE ACQUISITION MANAGEMENT

O MAJOR ACTIVITIES

- SPECIFYING CONTRACTUAL REQUIREMENTS
- PREPARING REQUESTS FOR PROPOSALS
- SOURCE SELECTION/NEGOTIATION
- REVIEWS AND AUDITS
- ACCEPTANCE TESTING AND INSTALLATION

O DISCIPLINES REQUIRED

- PROGRAM MANAGEMENT
- SYSTEM AND SOFTWARE ENGINEERING
- CONTRACT MANAGEMENT
- TEST AND EVALUATION
- COST MANAGEMENT
- LOGISTICS

What is NASA's primary Space Station software management challenge? It's not building software in house as in the past, it's not developing new software technologies or, in short, solving a traditional NASA engineering problem. These are all important, but not the real problem.

The primary challenge is to develop effective means for NASA to manage the development of software by contractors on a massive and geographically dispersed basis. This will also include the management of hundreds of subcontractors.

Therefore, the activities that NASA management must be primarily concerned with are the activities shown here. This requires a multiplicity of disciplines, most of which are not software engineering per se.

SUMMARY VIEW OF SOFTWARE MANAGEMENT ISSUES (CONTINUED)

NASA SOFTWARE ACQUISITION CHALLENGES

- O ESTABLISHING TECHNICAL AND HUMAN PERFORMANCE REQUIREMENTS
- O ESTABLISHING CRITERIA FOR SOFTWARE DESIGN VERIFICATION
- O ESTABLISHING CRITERIA FOR SOFTWARE ACCEPTANCE
- O CONTROLLING SOFTWARE ACQUISITION COSTS AND SCHEDULES
- O MINIMIZING DECISION CYCLE TIMES
- O PROMOTING AND ENFORCING SOFTWARE ENGINEERING PRACTICES
- O CONTRACTUALLY SUPPLYING TOOLS TO CONTRACTORS
- O DEALING WITH POOR CONTRACTOR PERFORMANCE
- O ESTABLISHING CONTRACTOR INCENTIVES
- O DEVELOPING A CRITICAL MASS OF SOFTWARE EXPERIENCED ACQUISITION PERSONNEL

In my opinion, these are NASA's primary software management challenges. Since software acquisition (not in-house development) is the central issue, NASA must undergo a rapid cultural change from a scientific and engineering oriented organization to become an astute buyer of software.

SUMMARY VIEW OF SOFTWARE MANAGEMENT ISSUES

(CONTINUED)

SPECIAL PROBLEM AREA

COST ACCOUNTING AND CONTROL

O TYPICALLY DIFFICULT FOR SOFTWARE CONTRACTORS TO COMPLY

- **EMPHASIS ON MANUFACTURING COSTS**
- **COST CENTER ORIENTATION RATHER THAN PRODUCT OR PROJECT**
- **NO SEPARATION OF HARDWARE AND SOFTWARE COSTS IN ENGINEERING ORGANIZATIONS**
- **LITTLE SOFTWARE HISTORICAL COST INFORMATION**

O BENEFITS FROM A WELL-DESIGNED (AND IMPOSED) COST SYSTEM

- **PROMOTION OF RESPONSIBILITY ACCOUNTING**
- **PROJECT AND LIFE CYCLE PHASE COST IDENTIFICATION**
- **COST AND SCHEDULE MORE PREDICTABLE (WHEN COUPLED WITH A PROJECT CONTROL SYSTEM)**
- **BASIS FOR METHOD AND TOOL IMPROVEMENT DECISIONS**

The essence of this special area is that most software contractors will be subcontracted to primes that build hardware systems. As a result, NASA will be managing software acquistions in the form of component parts of larger systems. This presents a major cost control challenge.

From NASA's perspective, it will be very difficult to gain insight into what is happening within contractor organizations unless special efforts are taken to develop and impose software cost accounting and control systems on the suppliers. This is a problem the Department of Defense has been grappling with for over a decade. NASA should take advantage of their lessons learned and current solutions through their STARS program interface to achieve the benefits shown above.

SUMMARY VIEW OF SOFTWARE MANAGEMENT ISSUES

BOTTOM LINE

ESSENTIAL REQUIREMENTS

- O TOP LEVEL PRODUCT PLAN (AND ASSOCIATED DOCUMENTATION AND FUNCTIONAL PLANS
  - DEFINE ACTIVITIES, SCHEDULES, RESPONSIBILITIES, DELIVERABLES
  - ADDRESS BUSINESS AND TECHNICAL ISSUES
- O PRODUCT LIFE CYCLE PROCESS FRAMEWORK
  - DISCRETE PHASES AND STEPS
  - EACH STEP COMPLETED BEFORE PROCEEDING (TO INCLUDE INTERATIONS FOR CORRECTIVE ACTIONS)
  - SOFTWARE INCLUDED IN EARLY SYSTEM PLANNING
- O MANAGEMENT PHASE REVIEW PROCESS
  - FORMAL CHECKPOINTS
  - CONSCIOUS DECISIONS
  - ESCALATION OF MANAGEMENT ISSUES
  - ACTIVE APPROVAL TO PROCEED

NASA must have a top level product plan which is deliverable oriented to identify the tangible items they are trying to acquire. The life cycle framework is required to form a basis for that approach and also a structured management review process to control contractor activities. All of this is used to ensure that timely decisions can be made to contain risks and keep Space Station plans on track.

This leads to my personal recommendations on the next page.

INITIAL RECOMMENDATIONS

<u>NASA SOFTWARE MANAGEMENT SHOULD:</u>

- ESTABLISH <u>PRODUCT</u> MANAGEMENT DISCIPLINE AS A STANDARD BUSINESS PRACTICE
- SYSTEMATICALLY BREAK DOWN WORK AND DEFINE EXPLICIT <u>WORK PACKAGES</u> WITH CRITERIA FOR THEIR SUCCESSFUL COMPLETION
- DESIGNATE SPECIFIC FUNCTIONAL AND WORK PACKAGE <u>RESPONSIBILITIES</u>
- PUT NECESSARY <u>RESOURCES</u> INTO PLACE TO CARRY OUT RESPONSIBILITIES
- PROVIDE MANAGERS WITH <u>AUTHORITY</u> TO CARRY OUT THEIR RESPONSIBILITIES
- ENSURE THAT <u>PHASE REVIEWS</u> ARE USED
- PARTICIPATE IN PHASE REVIEWS AND TAKE <u>PERSONAL RESPONSIBILITY FOR THEIR RESULTS</u>
- TAKE TIMELY <u>CORRECTIVE ACTIONS</u> TO MEET OBJECTIVES

ISSUES AND RECOMMENDED ACTIONS

1. ISSUE: Level A/B Software Management Plan

The draft Level A/B Software Management Plan (SMP) does not address several items either at all or with the proper emphasis.

RECOMMENDED ACTION:

The structure of the Software Management Plan should be modified to provide an easily identifiable place for all the issues to be addressed and given the proper emphasis. Table 1 contains the recommended Table of Contents for the Level A/B Software Management Plan, produced by panel consensus. Table 2 contains the recommended Table of Contents submitted by Robert Braslau of TRW without the benefit of the other panel members' review and comment. The panel recommends that the Level A/B Software Management Plan be modified and rewritten following the Table of Contents provided in Table 1.

IMPACTS REVISED SMP SECTIONS: All

2. ISSUE: Interdisciplinary Interfaces

The Space Station is a large, complex system composed of many subsystems. It is important that the relationships of software to the subsystems, overall system, and other disciplines, such as ground users, be well defined, and that control mechanisms and responsibilities be developed.

RECOMMENDED ACTION:

A program this large and complex must have well-defined interfaces and control mechanizations which should be explicitly identified in the Software Management Plan.

IMPACTS REVISED SMP SECTION: 3.2

3. ISSUE: Software Inheritance

There is a major opportunity to significantly reduce cost and increase reliability of Space Station software if existing NASA software can be reused or modified. Even use of existing, proven software design documentation is more cost effective when the actual software itself is impractical to transport directly. Obviously, many considerations will impact the practicality of such reuse.

New computers and a new language, among other considerations, will certainly complicate the issue. However, with no policy, it is clear that even an attempt at salvage will likely not occur.

In reviewing potential applications, it is probable that the highest likelihood for reusability will occur at the ends of the spectrum - major systems like mission control and orbit determination - or at the subroutine level, usually in standard support functions or specific algorithms.

Additionally, if a common language is used for Space Station development, opportunities should be examined even among new applications to see if potential redundancy can be eliminated by better organization and planning of acquisitions. As a final, obvious point, commercial software packages could be the most cost effective way of all IF they apply and are validated, and if the support and proprietary considerations can be worked out.

RECOMMENDED ACTION:

The Software Management Plan should address the reuse, inheritance, and co-existence with existing software. A policy should encourage the maximum reuse of existing software through cost trade-offs of requirements and design involving current capabilities, programs, and facilities; the use of commercial vendor supported products when appropriate; and the definition of interfaces to preserve current interfaces to permit continued joint use of established space data systems and communications as an option. Waivers to documentation requirements would be permitted where supplements to existing documents would suffice for slightly modified or commercial products. Software standards should be written to encourage the future reuse of software modules. Existing routines and tools should be selected and collected into a Space Station program-wide library with easy access and related support.

IMPACTS REVISED SMP SECTION: 2.10

4. ISSUE: Cost/Schedule/Technical Controls

The ability to control a software effort of the size and magnitude of the Space Station requires management to establish a measurement system to allow it to relate technical progress to cost and schedule performance throughout the developmental life cycle. The measurement system, once established, would provide managers with the ability to status where they are and determine what resources it would take to realize their plans. The measurement system would provide managers with timely visibility into actual performance using a combination of proven, earned-value, and variance reporting techniques. Technical performance measures would be established, tracked, and reported as a means to assess trends and reduce risk.

RECOMMENDED ACTION:

The Software Management Plan should specify policies and procedures for controlling cost, schedule, and technical performance of the software effort.

IMPACTS REVISED SMP SECTIONS: 2.11, 5.1, 5.2, 7.0

5. ISSUE: Risk Management

The Software Management Plan does not address the management of RISK. There are no policies, procedures, or provisions for the identification, reporting, controlling, resolving, or avoidance of risk items.

RECOMMENDED ACTION:

The Software Management Plan should be modified to include policies and procedures for proper planning, early detection, and resolution (risk avoidance), as well as for the identification, reporting, controlling, and resolution of risk items. There should be a top level policy on the establishment and utilization of reserves (dollars, staff, schedule, facilities, and other required resources).

IMPACTS REVISED SMP SECTIONS: 2.6, 10.0

6. ISSUE: Technical Performance Measurement (TPM)

The Software Management Plan does not specify any policies or procedures for acquiring/developing software that is designed and constructed in a cost-effective manner or that meets the required technical performance of the Space Station system.

RECOMMENDED ACTION:

The Software Management Plan should specify the policies and procedures for establishing technical performance items (e.g., software execution time, precision, memory usage, CPU utilization, storage utilization, response time, etc.), their measurement, reporting of actuals versus requirements, and resolution of nonconformance. The policies and procedures should address acquisition practices for establishing contract incentives that will highly motivate contractors to meet specified technical performance requirements.

IMPACTS REVISED SMP SECTIONS: 2.12, 5.2

7. ISSUE: Software Engineering

The procurement policies need to be expanded and detailed regarding contractor adherence to established software engineering (software design, coding and verification, principles and procedures). Specific software engineering principles and practices should be specified.

RECOMMENDED ACTION:

The Software Management Plan should emphasize quality standards consistent with the software category which are derived from criticality of use and potential consequences of errors. Software policies should be flexible enough to accommodate new paradigms as they become accepted industry practice. The policies should encourage the use of mathematically based logical deduction for the requirements and design verification of critical software kernels. Use of prototyping and evolutionary development methods as well as design language based software descriptions should be permitted. The state of software engineering should be reassessed periodically throughout the Space Station's existance to encourage the use of the most advanced practices and discourage obsolete practices, where operationally viable and cost effective.

IMPACTS REVISED SMP SECTIONS: 2.20, 4.3

8. ISSUE: Software Maintainability

It is well established that the cost of maintaining (evolving) software during continuing operations far exceeds the original development cost. Further, the planning required to both adequately prepare for the maintenance phase and ensure that the developed product is built with maintainability characteristics in mind must be accomplished before the actual development is initiated.

Because of the projected long life of the Space Station Support Systems, including software, the issue of software sustaining engineering (maintenance) must be considered during the planning and acquisition phases. To accomplish this, two aspects of software maintainability must be included in the Software Management Plan proper policy regarding the consideration of software maintainability characteristics during acquisition.

a. The acquiring agency for the software should be required to prepare a Software Support Plan prior to implementing acquisition activities. This plan will include the projected plans and requirements for post-development support of the software to be acquired. It will discuss the projected support strategy, the need for special tools and facilities during the sustaining engineering phase and the restrictions or requirements to which the developing organization must adhere to assure the most cost effective and efficient post-development maintenance and evolution of the product. Inclusion of these characteristics in a Software Development Standard or guidebook which could be extracted and tailored to the needs of a specific implementation might be the most effective method to achieve uniformity and completeness.

b. During acquisition, the acquiring agency must consider and include as requirements in their specification those elements of "built-in" software maintainability deemed critical to the product.

RECOMMENDED ACTION:

The Software Management Plan should have a section on software maintainability issues. This section should require that a Software Support Plan be developed and approved prior to initiation of acquisition activities. This plan should define the planning and projected requirements for post-development support of the proposed software and should provide guidance to the acquiring organization on the maintainability characteristics to be included during product development.

IMPACTS REVISED SMP SECTIONS: 2.7, 6.2

9. ISSUE: Independent Verification and Validation

An independent verification and validation (IV&V) organization to objectively assess the technical integrity of developer products continuously throughout the software development process should be selectively used to minimize the cost and maximize the effectiveness of the activity. By focusing on criticality, Space Station management can direct the attention of the IV&V organization to the areas where they get the largest return on their investment.

RECOMMENDED ACTION:

The policies on IV&V in the Software Management Plan should be tailored to selective use arising from criticality criteria.

IMPACTS REVISED SMP SECTIONS: 2.9, 7.0, 8.0

10. ISSUE: Firmware

The applicability of the Software Management plan to all forms of "firmware" needs to be specified, both for software engineering issues and for software management procedures.

RECOMMENDED ACTION:

The Software Management Plan should establish development, production, and maintenance policies addressing firmware. These policies should acknowledge and handle both permanent and modifiable PROMS. Newly developed or modified firmware should be treated as software until qualification or acceptance, and treated as hardware thereafter. The software support environment should include the tools to support firmware. Configuration management should include the handling of firmware, and documentation should be maintained to describe its design based on the degree of criticality of the embedded component.

IMPACTS REVISED SMP SECTIONS: 2.14, 4.4

11. ISSUE: Software Quality

The Software Management Plan should address modern approaches, focusing on quality as part of the procurement process, and should define the contract development and NASA procedures for focusing on early statistical assessment of software "goodness". The benefits of early attention to good software engineering are very significant in a long-life-cycle system (30 years).

RECOMMENDED ACTION:

Emphasize software quality in new paradigms made possible by new technologies. Define procurement policies for software development under statistical quality control using mathematics-based software engineering. Expand IV&V technology to provide statistical quality measurements of software, including certified estimates of mean time to failure (MTTF) and expected corrections required (ECR) for the life of delivered software products. Use IV&V in incremental development to provide early estimates of software quality and to permit corrective action in software development where required. Continuously assess new opportunities in software technology to procure higher quality software.

IMPACTS REVISED SMP SECTIONS: 2.5, 7.0

12. ISSUE: Mainstream Integration

The current NASA concern for highlighting and emphasizing software issues during Space Station development is correct and is key to successful Space Station implementation. However, care must be exercised to ensure that this increased concern for software does not destroy, conflict with, or interfere with the management of the system context in which the software must operate.

RECOMMENDED ACTION:

1. Ensure that system specifications are complete in the systems context, including both hardware and software implications.

2. Maintain consolidated configuration control of the baselined system specification and ensure that software changes are reviewed by the control board responsible for system specification integrity.

3. Maintain consolidated interface control for the total evolving system, including software.

4. During product (system) integration, ensure that the software developers are contractually required to support their product.

5. Provide for a single authority during system testing who has management control over all elements being integrated, including software, to ensure responsive action to anomaly detection, isolation, and correction.

IMPACTS REVISED SMP SECTIONS: 1.0, 3.1

13. ISSUE: Tailoring

The Space Station will produce many different types of software, each with a different life cycle, during the course of the project. To minimize cost and maximize development control, provisions are needed that allow software managers to tailor the policies of the Software Management Plan to specifics at hand. For example, documentation required for on-board systems may be different than that required for factory test equipment, especially if it is never delivered to NASA.

RECOMMENDED ACTION:

Define different categories of software and their life cycle and develop tailoring criteria that allow the Software Management Plan to be applied in a manner that minimizes cost and risk of development.

IMPACTS REVISED SMP SECTIONS: 2.1, 2.3, 2.21, 4.4

14. ISSUE: Review Process

The Software Management Plan should be more specific regarding the procedures for formal reviews. On a large program like Space Station, the quality of the reviews translates into the quality of the product and the risk metric.

RECOMMENDED ACTION:

Specific policies should be included in the Software Management Plan covering the formal software design and readiness review process. Each software review policy should address prerequisite preparation activities, the data package contents, the ojectives of the review, the attendees' responsibilities, and the relationships and timing relative to the associated system level reviews. The policies should also provide guidance and ensure that feedback on the review process itself is gathered and evaluated to determine how to improve its effectiveness.

A candidate set of formal software reviews includes:

- Operational Concept Review
- Software Requirements Review
- Preliminary Design Review
- Detailed Design Review
- Test Readiness Review
- Acceptance Test Review
- Launch Readiness Review
- Operations Readiness Review

IMPACTS REVISED SMP SECTIONS: 2.8, 4.2, 5.3

15. ISSUE: Incentives

The Software Management Plan should contain a policy encouraging incentive-type contracts based upon software quality metrics.

RECOMMENDED ACTION:

Software Management Plan should encourage the use of contractual incentives as a means of ensuring the quality and timeliness of software development and maintenance. The criteria for incentive determination should be objective, easy to understand, quantitative, and based on desired objectives, such as operational technical performance, quality, productivity, cost of ownership and timeliness. Incentive awards should be scheduled at predetermined intervals throughout the contract period of performance.

IMPACTS REVISED SMP SECTIONS: 8.0

16. ISSUE: Acquisition versus Development Management

Although it is expected that the majority of software to be utilized in the Space Station Program will be acquired from other organizations, some software such as simulations and testing tools will be developed in-house. Major aspects of these two processes are sufficiently different to warrant specific and clearly separated policies and guidance. Software acquisition management, for example, must be particularly concerned with procurement. Important aspects include the clear and complete specification of the product attributes and the acceptance tests that will prove that the product meets those attributes. Software development management, on the other hand, must more specifically address design and coding techniques, unit and integration testing, and development reviews.

RECOMMENDED ACTION:

NASA should clearly delineate policies and guidelines specific to software acquisition management and those applicable to software development management. No confusion should result for the manager attempting to determine the policies and guidelines that apply to each particular situation.

IMPACTS REVISED SMP SECTIONS: 1.0

17. ISSUE: Software Standards

Both industry and government have spent many years and work hours in developing software standards. None is perfect, but they are adequate. They are all based on a standard model. There seems little reason to "reinvent" a new standard.

RECOMMENDED ACTION:

Adopt software standards from either government (ref. 2) or industry (IEEE or other) and concentrate efforts more on products - their quality and acquisition.

IMPACTS REVISED SMP SECTIONS: Appendix

18. ISSUE: Life Cycle Process

The Space Station project needs to consider software throughout the system development process so that its effects on technical performance and life cycle cost can be thoroughly evaluated. Systems engineering activities should be augmented so that the software ramifications of early systems design and requirements engineering decisions can be ascertained and traded off. Operations and sustaining engineering aspects of software should be included in the process framework so that their implications can be assessed early and true life cycle analysis and cost trade-offs can be conducted. The hardware, software, and firmware life cycle processes should be interrelated across multiple life cycle horizons so that requirements are allocated properly and systems are reliable, maintainable, and available as needed.

RECOMMENDED ACTION:

The life cycle definition should be extended in scope to encompass systems engineering, subsystem development and operations, and sustaining engineering. The relationships between the hardware, software, and firmware life cycles need to be defined as do the products associated with the life cycle events.

IMPACTS REVISED SMP SECTIONS: 2.21, 4.2

19. ISSUE: Relationships to Non-Space Station Projects

The relationships and interfaces with interacting but separate projects from Space Station should be clearly identified and addressed in the Software Management Plan. Each relationship should be controlled by a Memorandum of Agreement covering responsibilities and operations, and the technical interface should be maintained in an Interface Control Document.

IMPACTS REVISED SMP SECTIONS: 3.2, 3.3

20. ISSUE: Management Tools/Environment

Management needs computer-based tools to assess project status, analyze risk, prepare schedules and budget, and evaluate cost/schedule/technical performance. These tools should mechanize methods established to provide managers with visibility and control and should allow managers to do their job quicker and better. A distributed management tool environment is needed that integrates financial, configuration management, library, and project management data in such a way that useful information flows out to the project manager. Existing tools and technology can be employed in such an environment to reduce development cost and speed up the implementation of an integrated NASA-wide management system for the Space Station Program.

RECOMMENDED ACTION:

The Software Management Plan should require that a software management environment be created to automate its policies and procedures across NASA centers.

IMPACTS REVISED SMP SECTIONS: 2.22, 4.3, 5.4

21. ISSUE: Change Control of Plan

It should be recognized that changes in the conduct of the Space Station Program will be necessary to incorporate lessons learned, exploit unexpected technology breakthoughs, deal with unforeseen difficulties, and recognize new management realities.

RECOMMENDED ACTION:

Provide explicit procedures in the Software Management Plan change as well as change control. Provide for continuous assessment and review of the Software Management Plan and define multilevel authorities for policy changes, permitting limited freedom for low-level changes that remain consistent with higher level policies.

IMPACTS REVISED SMP SECTIONS: 1.2

22. ISSUE: International Participation

The European Space Agency, the National Space Development Agency of Japan, and Canada have accepted President Reagan's invitation to participate in the development and subsequent operation of the Space Station. It is anticipated that the respective partners will utilize a significant portion of common software (such as for overall integration and checkout) and will jointly use the resulting in-space as well as ground facilities to conduct operations of common or individual interest. It is therefore very important that substantial commonality and standardization exist in the guidelines by which the software is acquired and maintained. This should include documentation types and formats, testing procedures, participation in major reviews, and exchange of pertinent status information.

RECOMMENDED ACTION:

The Space Station Program should strive to define areas requiring common and/or standard software management policies, plans, procedures, and standards. Management and technical interfaces should be indentified and defined as soon as possible. The Program should coordinate with its foreign partners to formulate, review, and then update on an ongoing basis the affected products and the management guidance. An important consideration in this activity will be undesirable technology transfer and protection of proprietary software techniques, tools, and products. The Space Station Program should work closely with its legal experts to define criteria and rules applicable to international considerations.

IMPACTS REVISED SMP SECTION: 3.4

23. ISSUE: Security

The Software Management Plan does not have sufficient emphasis on the policies and procedures for proper handling of data and specification of system design as necessary to meet the requirements of system and data security, privacy, sensitivity, and safekeeping.

RECOMMENDED ACTION:

The Software Management Plan should be modified to include the policies and procedures that address the data handling and system design requirements to ensure that the project needs, reasonable and prudent safeguards, civil laws, and government regulations are properly addressed in the acquisitions/development and operation of the computer-based systems, particularly in the software.

IMPACTS REVISED SMP SECTIONS: 2.19, 9.0

24. ISSUE: Timely Decision Making

The Space Station approach and procedures for making critical decisions should be specified. Where the risk is appropriate, specify the decision authority as low in the management structure as possible.

RECOMMENDED ACTION:

Define the policy making decision process and the levels and authorities for defining policy. Provide for low-level flexibility in policy definition and change that is consistent with upper-level policy. Schedule and publish critical decision points with wide and long-range effects, and provide time and opportunity for interested parties to offer opinion in the decision process. Set up a program outside normal management structure to receive suggestions and criticisms of policy with appropriate rewards as well as investigative and reporting facilities.

IMPACTS REVISED SMP SECTIONS: 1.0, 2.11, 5.4

25. ISSUE: Continuous Operations Contingency

The Software Management Plan does not call out the proper policies and procedures for ensuring that there is very low probability of the loss of correct data and/or software during acquisition/development and operations.

RECOMMENDED ACTION

The Software Management Plan should be changed to specifically address the policies and procedures to ensure that both NASA in-house staff and contractors acquire/develop and use software following practices that will have a very low probability of loss of software or data and will have the ability to modify or automatically regenerate executable software and operational data.

IMPACTS REVISED SMP SECTIONS: 2.7, 9.0, 10.0

26. ISSUE: Product Orientation

The orientation of the Space Station Program is towards the acquisition of products rather than their development.

RECOMMENDED ACTION:

The Software Management Plan should focus on the acquisition of software rather than software development, and with more of a product orientation; i.e., it should address the control, quality, and management of PRODUCTS rather than of the process by which they are to be produced. The Software Management Plan should provide policies and guidance for the acquisition process.

IMPACTS REVISED SMP SECTION: 1.0

27. ISSUE: Design-To-Cost

A Design-to-Cost concept for the entire Space Station Program should be promulgated and clarified in the Software Management Plan. Software policies should permit the identification of critical requirements significantly affecting system, subsystem, or software development/operational costs. A methodology and associated analysis concepts and tools should be adopted for prioritizing requirements, encouraging cost benefit analysis, and providing the operational flexibility to adjust to the resulting constraints necessary to live within predefined cost budgets.

RECOMMENDED ACTION:

Design-to-cost should be defined and promulgated as one potential contracting vehicle when under severe budget constraints with requirements that contain the potentiality for trade-off (e.g., you are willing to settle for as much as you can get for a set price). It will be extremely important to review the selection of design-to-cost procurements prior to execution to assure the items being procured are really amenable to this form of contracting as opposed to normal practices with extremely rigid contract management.

IMPACTS REVISED SMP SECTIONS: 2.13, 5.1

28. ISSUE: Goal Setting and Clearly Stated Objectives

The Space Station Program is to be commended for placing high priority on the early identification and formulation of overall software managment policies and guidance. However, a critical component of that thinking must be the clear and comprehensive statement of Space Station Program goals and objectives relative to software. These goals and objectives should be in consonance with the overall program goals and objectives and should be specific enough that criteria can be established to ascertain attainment.

RECOMMENDED ACTION:

The existing draft of the top-most Software Management Plan should be revised to clearly state the plan's purpose and to specify the overall goals and objectives to be accomplished by Space Station software. These goals and objectives should cover both strategic and tactical considerations.

IMPACTS REVISED SMP SECTION: 1.0

29. ISSUE: Lessons Learned

The value of learning from past software efforts is increasingly being recognized as a valuable way to avoid repeating mistakes and encountering pitfalls. Information such as software costing estimates versus actuals as a function of costing technique and life cycle phase, staffing levels and types versus acquisition performance, and true capabilities of testing tools and techniques is very helpful, particularly to long-term programs with much software maintenance and enhancement. Such data is not collected without cost, however. Resources must be dedicated to the tasks of collecting, filtering, organizing, and analyzing the lessons learned information.

RECOMMENDED ACTION:

The Space Station Program has a very long expected lifetime. Its software will be continuously enhanced and changed as new requirements are brought forward. Personnel will change. Minimization of long-term costs virtually mandates that the program intentionally monitor itself and learn from past experiences. The Space Station Program should establish mechanisms for capturing lessons learned and improving procedures to make maximum use of such lessons. It is suggested that one relatively easy way to gather such data is as part of each major review.

IMPACTS REVISED SMP SECTION: 2.16

30. ISSUE: Standardization Process

The Space Station Program will involve the development of many diverse subsystems by different NASA centers and contractors. It is important that policies be established to standardize how software is procured. Such issues as multiple licensing agreements, maintenance clauses, delivery standards, documentation, and product standards need to be addressed.

RECOMMENDED ACTION:

The Software Management Plan should provide policies, procedures, and guidance to ensure an appropriate level of standardization across the Space Station Program.

Similar procurement procedures and management controls must be used throughout the program.

IMPACTS REVISED SMP SECTIONS: 2.15, 4.0, 5.0, 6.0, 7.0, 8.0

TABLE 1

SPACE STATION LEVEL A/B SOFTWARE MANAGEMENT PLAN

RECOMMENDED TABLE OF CONTENTS

4.4 Tailoring
4.5 Deviations & Waivers

5.0 Management Controls

5.1 Cost and Schedule Controls
5.2 Technical Performance Measurement
5.3 Management Reviews & Reporting
5.4 Technical Management Information Systems
5.5 Administrative Controls

6.0 Configuration Management

6.1 Evolution
6.2 Maintainability

7.0 Quality Management

8.0 Procurement Approaches

9.0 Security

10.0 Risk Management

10.1 New Technologies
10.2 Disaster Recovery
10.3 Reserves

APPENDIX A. Space Station Software Segments
B. Software Support Environment/TMIS
C. Standards

NOTE: Jody Steinbacher recommends that the Policy section be organized so that related policies are together, for example: 2.21, 2.20, 2.3, 2.1, 2.14, 2.10, 2.9, 2.14, and possibly 2.15; and 2.2, 2.4, 2.5, 2.6, 2.7, 2.22, 2.8, and 2.23; and 2.11, 2.12, 2.13 and 2.17; etc.

TABLE 2

RECOMMENDED REORGANIZATION/OUTLINE OF THE LEVEL A/B SOFTWARE MANANGEMENT PLAN BY ROBERT BRASLAU, TRW

1.0 PURPOSE AND SCOPE

- 1.1 Level A Charter
- 1.2 Management Plan Maintenance
- 1.3 Scope of the Space Station Software
- 1.4. Overall Software Development and Operational Objectives
- 1.5 Related Software Standards
- 1.6 Applicable Documents

2.0 ORGANIZATION AND RESPONSIBILITIES

- 2.1 Program Structure and Software Responsibilities
- 2.2 Review Boards and Advisory Panels
- 2.3 Interface Control Working Groups
- 2.4 System Engineering and Integration

3.0 SOFTWARE POLICIES

- 3.1 Level C and Contractor Software Management Plans
- 3.2 Operational Concepts Definition
- 3.3 Operational Concepts Readiness Review
- 3.4 Requirements and Interface Specifications*
- 3.5 Software Requirements Review*
- 3.6 Preliminary Design Specification*
- 3.7 Preliminary Design Review*
- 3.8 Detailed Design Specification*
- 3.9 Detailed Design Review*
- 3.10 Structural Software Design*
- 3.11 Unit Development Folders*
- 3.12 Design Walk-throughs*
- 3.13 Implementation Program Standards*
- 3.14 Unit Test Planning and Testing*
- 3.15 Software System Integration and Test*
- 3.16 Acceptance Test Plan and Procedures*
- 3.17 Data Generation and System Build
- 3.18 Adaptation and Mission/Payload Data Management
- 3.19 Test Readiness Review
- 3.20 cceptance Test Review and Delivery
- 3.21 Launch Readiness Review
- 3.22 Operation and Maintenance Products
- 3.23 Operations Readiness Review
- 3.24 Controlled Documentation and Products*
- 3.25 Configuration Management*
- 3.26 Quality and Integrity Management*
- 3.27 Uniform Development Environment*
- 3.28 Management Information System
- 3.29 Metrics and Experience Collection*

* TRW has 1-2 page policies that could be used as models for Space Station.

3.30 Risk Management
3.31 Independent Verification and Validation
3.32 Software Reuse, Inheritance and Coexistance
3.33 Performance Measurement and Status System
3.34 Technical Performance Production and Measurement
3.35 Design-to-Cost Methodology
3.36 Firmware
3.37 Standardization
3.38 Contractor Incentives
3.39 Privacy and Security Protection

4.0 LIFE CYCLE PROCESS MANAGEMENT

4.1 Phases, Baselines, Milestones
4.2 Design and Readiness Reviews
4.3 Overall Development Schedule and Milestones
4.4 Software and System Integration and Test
4.5 Operations Maintenance
4.6 Technology Insertion and Space Station Evolution

5.0 MANAGEMENT CONTROLS

5.1 Management Status Review and Reporting
5.2 Cost and Schedule Performance Controls
5.3 Technical Performance and Operations Resource Monitoring
5.4 Software Support Environment Usage
5.5 Technical and Management Information System Usage
5.6 Acceptance of Deliveries and Software Ownership
5.7 Data Generation and Verification Management
5.8 Subcontractor Monitoring
5.9 International Interface Change Impact Management
5.10 Operations Conflict Resolution
5.11 Interface Management
5.12 Technical Review Boards and Advisory Panels Operation

6.0 CONFIGURATION MANAGEMENT

6.1 Baseline Definition and Control
6.2 Change Management
6.3 Software Library
6.4 Evolution and Maintainability

7.0 QUALITY AND INTEGRITY MANAGEMENT

7.1 Software Criticality Classification
7.2 Performance Factors and Metrics
7.3 Resource Utilization Monitoring
7.4 Problem Reporting and Close-out
7.5 Corrective Action System
7.6 Deviations and Waivers
7.7 Software Reliability/Availability and Safety

SOFTWARE DEVELOPMENT ENVIRONMENT PANEL SUMMARY

The Software Development Environment (SDE) Panel addressed key programmatic, scope, and structural issues raised by its members and the general audience regarding the proposed software development environment for the Space Station program. The general team approach taken by this group led to a consensus on 18 recommendations to NASA management regarding the acquisition and definition of the SDE. This approach was keyed by the initial issues presentation given by Barry Boehm to the general audience on the first day. Additional issues (for a total of 23) were developed by the panelists in their first closed session from which key areas were selected and discussed in open session. These discussions led to the following key recommendations summarized in the following table and described in the following text.

Key Recommendations

Programmatic	Develop uniform, NASA-furnished SDE; mandate compatibility with delivered software, do not mandate for development
	Develop SDE operations concept; use JSSEE as a starting point; use input from Phase B contractors and operational users
	Develop incrementally using identified guidelines
SDE Scope	Focus on products; non-prescriptive of detailed methodology
	Design to support software reuse
SDE Structure	Furnish as portable software package, except where requirements dictate hardware
	Virtualize the operating system; start with UNIX, prepare to evolve
	Establish a single subsetable SDE host; allow for multiple target support subsystems; maximize commonality; accommodate user-unique services
	Use a modular, layered architecture
	Instrument for self-diagnosis

Programmatics: The panel and audience strongly endorsed the concept of a uniform, NASA-furnished, mandated SDE to address the critical life-cycle cost and integration issues of Space Station software. Risks, such as schedule, technological obsolescence, and contractor incompatibilities, are mitigated by the following: an operations concept which provides for contractor options to use their own SDEs, as long as the delivered software is supportable by the NASA SDE; an incremental acquisition strategy; and the use of layered architectures to assure technological transparency.

A major recommendation which will mitigate schedule and product risk is to develop an SDE Operations Concept as soon as possible which addresses user requirements and lifecycle scenarios based on inputs from users, Phase B contractors, and similar DoD efforts (e.g., the JSSEE Operational Concept Document).

Scope: A key concern in this area is the degree of mandated software engineering methodology implied by the SDE. The panel strongly endorsed the concept that the SDE focus on products (such as specifications, design/code representations, etc.) rather than the methods, thereby allowing for contractor-unique approaches and new methods technology.

Another major aspect of the SDE scope strongly endorsed is the concept of a support library of reusable components, which could lead to a major savings in overall Space Station life cycle costs.

Structure: The key concern addressed is the architecture--modularized and layered--to allow for technological evolution at distinct levels. An approach was developed and presented for the critical interfaces to protect against predictable sources of change.

The major sources of SDE change and their corresponding information-hiding interfaces are:

Source of Change	Info-hiding Interface
o Text-processing Capabilities	o Text Files
o Requirements, Design, Code Representations	o Standardized Content at Each Stage
o Financial Management Capabilities	o Standard WBS
o DBMS Capabilities	o Abstract DBMS Interface
o Workstation Capabilities	o Abstract Workstation Interface
o CPU	o UNIX

Another major aspect of the SDE structure endorsed is that it consists of a subsetable set of tools engineered with uniform interfaces providing the SDE capability to customize to specific user requirements either by application (e.g., flight or ground software development, analysis, management, simulation), by type of user (e.g., expert/novice, specialist/generalist), or by type of equipment (e.g., mainframe, mini, or workstation).

RECOMMENDATIONS

1. THE Software Development Environment (SDE) should be a uniform, NASA-furnished, "mandated" environment supporting the use of existing NASA facilities.

2. The SDE should be furnished as a portable software package (except where requirements dictate hardware).

3. The SDE should have a virtualized operating system. Start with UNIX and prepare to evolve.

4. In order to maximize the commonality, the SDE should reside on a single host subsystem (where subsets of that host are possible and can support SDE subsets). The SDE should allow for multiple target support subsystems.

5. The SDE should be incrementally developed.

6. Consideration should be given to having an "SDE Flyoff" with multiple vendors, although the panel thought this may not be necessary.

7. The SDE application should be product oriented, not necessarily process oriented.

8. There must be a specific development and application plan along with a marketing program for selling to NASA Centers and vendors.

9. The SDE should be instrumented for self diagnosis.

10. The SDE must support software reuse.

11. An operations concept must be generated as soon as possible. Use the JSSEE (Joint Services Software Engineering Environment) operational concept as strong input. Also obtain inputs from the Phase B contractors and potential users.

12. Prototype the user interface early.

13. Collect and incorporate lessons learned from past NASA projects.

14. Any new software written for the SDE should be written in the chosen NASA space station programming language.

15. NASA should establish research activities to fill in the SDE gaps, i.e., develop new software environment technology where it is needed.

16. The SDE should have a modular, layered architecture.

17. NASA should define the criteria for SDE acquisition.

18. The SDE is to support reuse of existing NASA facilities.

D3

N86-23317

A VIEW OF SOFTWARE DEVELOPMENT ENVIRONMENT ISSUES

Barry Boehm
TRW

C-2

OUTLINE

- Nature of the challenge
- Orange-Book issues (ref. 1)
  - Pros, cons, assessment
- Additional SDE issues
  - DOD coordination
  - Scope of SDE
  - Reuse support
- Summary

NATURE OF THE CHALLENGE

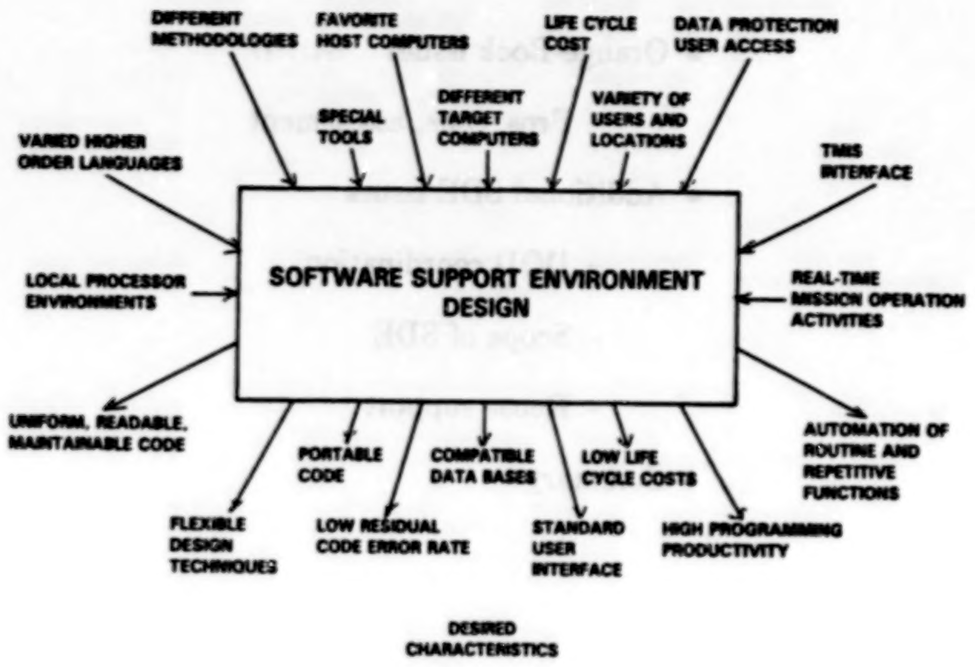
The SSDS is a mission support system for:

- Thousands of operators and decision makers
- At on-line terminals
- At many geographical locations
- Performing complex, interacting functions
- With imprecisely defined requirements
- In a dynamic, less-than-predictable environment
- Requiring essentially error-free performance

It is essential for coordinating the mission

It requires significant investments in time, dollars, talent

SDE Design Considerations



DIFFERENT METHODOLOGIES
FAVORITE HOST COMPUTERS
LIFE CYCLE COST
DATA PROTECTION USER ACCESS
SPECIAL TOOLS
DIFFERENT TARGET COMPUTERS
VARIETY OF USERS AND LOCATIONS
VARIED HIGHER ORDER LANGUAGES
TMIS INTERFACE
LOCAL PROCESSOR ENVIRONMENTS
SOFTWARE SUPPORT ENVIRONMENT DESIGN
REAL-TIME MISSION OPERATION ACTIVITIES
UNIFORM, READABLE, MAINTAINABLE CODE
PORTABLE CODE
COMPATIBLE DATA BASES
LOW LIFE CYCLE COSTS
AUTOMATION OF ROUTINE AND REPETITIVE FUNCTIONS
FLEXIBLE DESIGN TECHNIQUES
LOW RESIDUAL CODE ERROR RATE
STANDARD USER INTERFACE
HIGH PROGRAMMING PRODUCTIVITY
DESIRED CHARACTERISTICS

ISSUES 1,2,3: UNIFORM, NASA-FURNISHED, MANDATED SDE

PRO:

- **Better software coordination**
  - **Fewer errors, interface problems**
- **Less duplication of effort**
- **Conceptual integrity**
  - **Reinforcement of management approach**
  - **SDE/user interface**
- **Controllability**
  - **Response to problems**
  - **Technology insertion**
- **Better life-cycle support**
  - **Ability to recompete maintenance**

ISSUES 1,2,3: UNIFORM, NASA-FURNISHED, MANDATED SDE

CON:

- Contractor incompatibilities
  - Competitive bias
- Technology insertion
  - Disincentives to experiment
- Implied SDE warranty
- SDE size, development risk
- Breadth of user community
  - Centers, contractors, researchers
  - Levels of expertise
  - Special functions: simulation, test, etc.
  - Large up-front training cost

ISSUES 1,2,3: UNIFORM, NASA-FURNISHED, MANDATED SDE

PRO	CON
•Better s/w coordination	•Contractor incompatibilities
•Less duplication	•Technology insertion
•Conceptual integrity	•Implied SDE warranty
•Controllability	•SDE size, development risk
•Life-cycle support	•Breadth of user community

ASSESSMENT

- Go for it – in ways which minimize cons
  - Pre-delivery contractor option to use own SDE
  - SDE modularized for technology insertion
  - Establish levels of warranty
  - Incremental development to reduce risk

ISSUE 4: PRECEDE SDE DEVELOPMENT WITH DEVELOPMENT OF FUNCTIONAL CAPABILITIES, PROTOTYPE, DETAILED SPECS

PRO	CON
•Familiar acquisition approach	•Very high schedule risk
•Provides criteria for choosing developer	•Not clear more prototypes will add much information

ASSESSMENT

- Better to go for early initial capability
- Use DOD JSSEE Spec as basis for defining requirements
- Run competitive flyoff for production - engineered initial SDE capability

ISSUE 5 BUILD LAYERED SDE

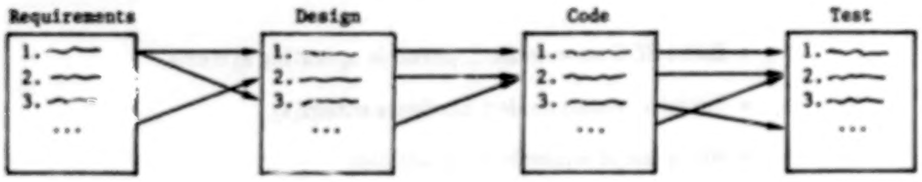
PRO	CON
•Accommodate change, growth, technology insertion	•Performance penalties
	•May pick wrong layers

ASSESSMENT

- Build layered SDE
  - Use info-hiding to modularize around major sources of change
    - Methodologies (requirements, design, management)
    - Mainframes, workstations
    - Networks, peripherals
    - Language, operating system?

MODULARIZING AROUND SOURCES OF CHANGE IN SOFTWARE METHODOLOGY

- **Make minimal assumptions on nature of elements (requirements, design, code, test, management)**
  - **Resolvable into separately identifiable items**
- **Develop traceability tool to track relations between items**



ISSUE 6: AFFILIATE WITH DOD ENVIRONMENT

PRO	CON
•Technical synergy	•Not clear which one
•Less contractor confusion	•Schedule mismatches
	•Control; coordination

ASSESSMENT

- **Propose coordinated, potentially joint SDE**
- **Volunteer to develop a pre-1990 initial SDE capability based on JSSEE spec**

ISSUE 7: FURNISH FULL-UP SDE:

SOFTWARE, CPU, WORKSTATION, LAN

PRO	CON
•Fewer coordination problems	•Expensive to furnish
	•Technology insertion problems
	•Contractor SDE incompatibilities

ASSESSMENT

- Build SDE on standard, portable operating systems
- Support recommended hardware subset(s)
- Allow use of equivalent capabilities

ISSUE 8: SUPPORT LIBRARY OF REUSABLE COMPONENTS

PRO	CON
•Major source of future s/w cost savings	•Added investment
	•Hidden incompatibilities
	•Component warranties
	•Version control
	•Component pollution

ASSESSMENT

- Go for it – in ways which minimize cons
  - Levels of warranty
  - Strong documentation, CM
  - Selective incorporation

SUMMARY

- **Building an SDE is in the same ballpark as building SSDS**
  - **Very complex, but essential**
- **Worth going for uniform, NASA-furnished, mandated SDE**
  - **In ways which minimize risks**
- **Value of further SDE prototyping unclear**
  - **Several de facto prototypes exist**
  - **Very high schedule risk**
- **Worth coordinating with DOD**
  - **JSSEE spec a useful starting point**
- **Furnish SDE as standard s/w on portable operating system**
  - **Support but not mandate CPU, LAN, workstation**

KEY ISSUES ADDRESSED

PROGRAMMATICS

Uniform, NASA-Furnished, Mandated SDE

The issue raised addresses the realization of SDE capabilities. Should NASA provide and require the use of a standardized SDE for Space Station software acquisition? Examination of this issue reveals considerations which require focused attention.

Uniformity will yield fewer interface and coordination problems and will provide conceptual integrity. These benefits, however, are at the expense of multi-contractor incompatibilities and their combined strengths for technological development.

A mandated, government-furnished SDE provides direct control by NASA for problem solutions, evolutionary as opposed to revolutionary growth (mature expansion), and more opportunity for SDE-related cost containments. However, any GFE item bears an implied warranty. This needs to be addressed by defining levels of warranty for components of the SDE. Another issue is how a government-furnished SDE would be sized to efficiently service the wide breadth of the anticipated user community. Here, the SDE needs to be organized to be easily subsetable to specialized user communities, host computers (maxis versus work stations), or user expertise levels.

SDE Operations Concept

The scope of SDE application is indeed broad. Each of the major workpackage contractors is likely to have unique, embedded software development methodologies and supporting facilities. In turn, their subsystem development organization and/or subcontractors will have established computer system development tools, experience, and expectations. Further, the ultimate users of the Space Station will include a significant portion of small groups or individuals interested only in their experiment or production package and not in any required supporting software. Effectively scoping the range of SDE requirements requires the near-term definition of how all users--big and small, sophisticated or naive, experienced or novice--may use the system. An Operations Concept, addressing how all users expect to use the system during its entire lifecycle, has been found extremely useful in establishing a basis for subsequent hardware/software requirements specification.

The conclusion reached gave an affirmative answer to the issue: NASA should provide and mandate the use of a uniform SDE. The government furnished SDE should be effected in a manner which mitigates benefits and risks, specifically by establishing a widely accepted SDE Operations Concept.

Incrementally Developed

If the SDE is constructed as a set of functional modules enclosed by a communications structure, the modules can be acquired, inserted, and replaced on an incremental schedule. The general driving requirement for module acquisition and insertion is at the communications interface. Initial priorities should be established by NASA so that incremental implementation will support program requirements as they become needed. Some, indeed, are needed now.

The SDE must be subsetable, modularized, and concentrically layered to assist all mission, management, and communication requirements. This form of structural detail seems most likely to be able to achieve the desired flexibility and versatility over the range of specific SDE instances.

A strategy for incremental development is recommended which minimizes the dependence of the SDE development schedule on requirements to be derived by Space Station Phase B contractors:

Increment 1: OS, DBMS, utilities, basic CM, office automation, and management functions

Increment 2: Basic requirements and design specification, planning and analysis support

Increment 3: Basic code, unit test, integration and test support

Increment 4: Basic real-time OS, DBMS, and utilities for flight and ground target computers

Increment 5, 6,...: User-prioritized additions and extensions to the above

This strategy allows NASA to get an early start on the portions of the SDE needed for initial Space Station program development support.

SDE SCOPE

Focus on Products

No clearly superior methodology for software design refinement has emerged, yet many have proven useful for unique or particular application arenas. For all methodologies, certain intermediate products or design representations are recognized. Focusing upon these products, as distinct from the methodology or process employed in establishing these products, permits considerable methodological flexibility and allows for future technology insertion. Where a generally agreed upon management model can be established, the SDE may support the process directly. We conclude that the SDE shall be nonprescriptive of a specific requirement or design methodology.

Supporting Software Reuse

Complete rebuilding of large software systems is no longer economically feasible. Full advantage must be taken of viable existing elements. Suitable reusable components may be commercially available off the shelf (COTS), may reside at one or more NASA centers, or may be adaptable from past contractor efforts. Making use of such elements requires careful initial attention to the framework or architecture of the SDE, including the definition of appropriate interfaces and the levels in the hierarchy. Clearly, multiple source languages and/or object code bodies should be accommodated in many instances. Certainly, the desired SDE subsetability considerations relate to the kind of structure promoting reuse described here.

We conclude that the SDE interface and architectural definitions should foster software reuse.

SDE STRUCTURE

Furnished as Portable Software Package

The SDE should consist of device-independent (loosely coupled hardware dependencies) functions such that changes in hardware do not have an effect on software functionality. Hardware availability should not drive the software requirements, but some well defined, vendor dependent elements may facilitate widespread use of currently available components. In some areas, such as target machine support, requirements may dictate a hardware component of the SDE.

Virtualized Operating System

The operating system which supports the SDE should be device and vendor independent insofar as possible. As a present starting point, UNIX appears to be the only candidate that meets this requirement and should be selected. Prevailing personal computer operating systems meet the spirit but not the large machine scope of this requirement. For the future, the SDE can implement other hardware-independent operating systems (e.g., CAIS or MAPSE for Ada) as they become available.

Single, Subsetable SDE Host

The central issue of the SDE structure is architecture. Associated subissues (incremental development, choice of modular or layered, ease of user accommodation) are facets of the SDE architecture issue perceived functionally as requirements.

Selection of the subsetable functions and interfaces is the most critical. A primary capability is to allow for support of multiple host targets. These subsetable functions must also support, by interface management, fully generalized and specific functions within the layered architecture. A major objective is to maximize commonality of widely used functions. There is a potential, as the SDE evolves over time, to yield unmanageable interface/function diversification. The result is that interfaces could multiply and become deeply nested, thus driving incremental mainframe costs of ownership for certain levels of capability.

The definitions of subsetable SDE elements, interface specification, communications/tasking network definition, and management provide the baseline from which to proceed. Plugability as to function, via the suitable interfaces, will result in achieving, integrating, and managing associated issues of portability, user interfaces, and mission requirements.

Instrumented for Self-Diagnosis

A rational basis for extension or improvement of the SDE can only come from an understanding of its strengths and deficiencies. Knowing how the SDE elements are employed by the spectrum of users throughout the life cycle of each particular software deliverable is a vital part of this understanding. We conclude that the SDE should automatically collect data that characterizes its use throughout the entire development process.

LANGUAGE PANEL SUMMARY

This panel was charged with making recommendations on the various language issues involved in the development of Space Station. This charge included the full set of development and user languages covering the entire life cycle of development and all types of user applications.

The selection and standardization of languages and interfaces for the Space Station program are critical needs to insure the success of this predominately engineering activity. While the Language Panel recognizes that the project life cycle will require a family of languages for the various classes of users and developers, it is crucial to begin making decisions which will focus planning efforts by limiting the range of possible selections. Requirements for the Space Station information system long-term maintenance and evolution will make it imperative that a high-order development language be utilized. It is recommended that the primary high-order language for source code development be Ada. (Ada is a registered trademark of the Department of Defense, Ada Joint Program Office.) Issues related to the utilization of Ada should be addressed as soon as possible. These include developing a transition strategy, providing education, accommodating the utilization of software already in existence, and developing fall-back options for high risk areas. One high-risk area is satisfying the requirements for run-time support for target systems, especially when the targets are distributed. Requirements for design specification languages or interfaces that complement Ada should be determined.

During its discussions, the panel operated under the basic assumption that Space Station is an engineering activity. Therefore, where appropriate, selection and standardization of languages and interfaces should begin constraining the degrees of freedom. The selection of languages and interfaces impacts the construction of a Software Development Environment (SDE), which is a substantially more critical component of Space Station software.

Although there were panels to discuss management, standards, environments, and languages, no panel was specifically charged with methodology issues. This is of real concern, and the language panel tried to address this issue whenever it was appropriate. The panel also felt that methodology should be discussed in any future meetings on software.

The panel was able by consensus to arrive at a total of 11 recommendations. These recommendations were discussed in the open forum, and there was felt to be reasonable agreement of the attendees at the open meeting.

These recommendations fall into 5 categories. Recommendation 1 deals with an important aspect of the whole software development process. Recommendations 2, 3, 4, and 5 deal with the choice of the software development language. Recommendations 6, 7 and 8 deal with languages at early phases of the life cycle. Recommendations 9 and 10 deal with user languages. The last recommendation says that NASA must track language technology in the future.

RECOMMENDATIONS

1. NASA should avoid premature commitment to hardware implementation decisions. System and software architecture should be defined first.

2. NASA should declare Ada now as the preferred high-order language for source code development and address the following issues as quickly as possible:

- transition strategy
- procurement issues
- interfaces to existing NASA software
- development of guidelines for applying Ada to various application areas
- development of appropriate run-time support environments for NASA applications
- education
- a liaison to DoD
- a seat on the Ada board
- benchmarks for performance
- prototyping
- development of appropriate tools to partition and allocate Ada entities across distributed applications
- introduction and utilization of reusable components
- investigation of fallback position options for high risk areas

3. The commitment to Ada requires an education program in software engineering methodologies with Ada, which should begin as soon as possible. The education includes the study of relevant examples. It should cover multiple levels of management, application programmers, etc.

4. NASA must define its requirements for the run-time support library and kernel for the target systems, including distributed targets.

5. NASA needs to define the requirements for the interface to the run-time system.

6. The first version of the SDE should not be constrained to have a single requirements language, AI expert systems language, or prototyping language.

7. NASA should determine the requirements for and select or develop requirements and design specification languages or interfaces that complement the SDE and Ada.

8. The design language should be syntactically and semantically consistent with the development language and should have on-line support for interface checks, etc.

9. For all levels of user interfaces, there should be a set of standards to provide commonality across all phases of the Space Station life cycle.

10. NASA should identify all categories of users and user interfaces, and quickly proceed with rapid prototyping to determine the real requirements.

11. Since Space Station software will evolve over 30 years, NASA should track language technology and act appropriately.

LANGUAGE ISSUES FOR SPACE STATION

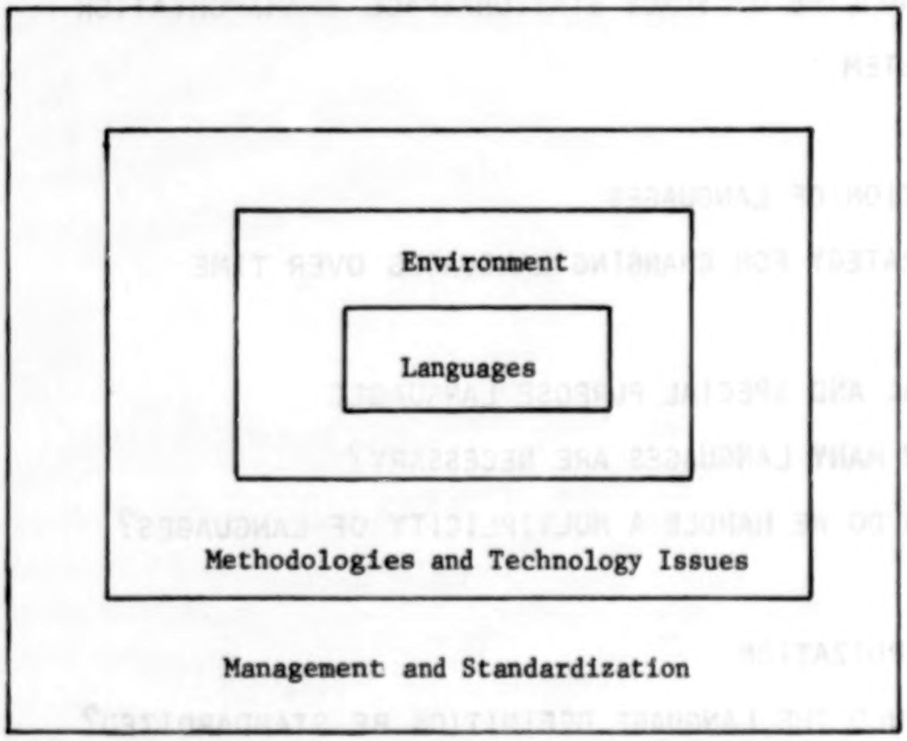
Professor Victor Basili began by reviewing the essential considerations and initial recommendations of 1984 workshop (ref. 1) given in the next three figures. He commented that language was to be considered as a notation and tool for supporting

- . application domains
- . phases of the life cycle
- . methods

in such a way that it satisfies criteria of ease of use, readability, efficiency, modifiability, portability, low cost, etc.

Therefore we need to (1) categorize applications, e.g. flight software, support systems, and operations, (2) categorize phases of the life cycle, e.g. requirements, design, code, and test, and (3) delineate methodologies and recommend languages or criteria for selecting a family or set of languages for use in Space Station.

One of the concerns in choosing languages is that because they are an integral part of the software development environment, the decisions on languages cannot be made independent of the decisions about that environment. In turn, the environment will and should be influenced and constrained by the methodological and technological issues decided upon for Space Station. These methodological issues will certainly be influenced by the management and standardization issues.


Environment
Languages
Methodologies and Technology Issues
Management and Standardization

ESSENTIAL CONSIDERATIONS

1. REQUIREMENTS

   HAVE DEFINED CANDIDATE LANGUAGES FOR OPERATION

   NEED STUDY FOR DEVELOPMENT

2. USE OF LANGUAGES

   COBOL, FORTRAN, HAL/S PRIMARY

   C, PASCAL, PL/1 SOME GAINS

3. SOFTWARE HERITAGE AND REUSABILITY

   LONG LIFE OF SPACE STATION-SPACE TRANSPORTATION SYSTEM

4. EVOLUTION OF LANGUAGES

   STRATEGY FOR CHANGING LANGUAGES OVER TIME

5. GENERAL AND SPECIAL PURPOSE LANGUAGES

   HOW MANY LANGUAGES ARE NECESSARY?

   HOW DO WE HANDLE A MULTIPLICITY OF LANGUAGES?

6. STANDARDIZATION

   SHOULD THE LANGUAGE DEFINITION BE STANDARDIZED?

ESSENTIAL CONSIDERATIONS

7. ASSEMBLY LANGUAGE

   HOW MUCH, IF ANY, ASSEMBLY LANGUAGE SHOULD BE ALLOWED?

8. TOOLS

   WHAT IS THE EFFECT OF LANGUAGE SELECTION ON AN INTEGRATED SET OF TOOLS?

9. MULTI-LINGUAL ENVIRONMENTS

   HOW ARE LANGUAGES CHOSEN TO BE COMPATIBLE WITH EACH OTHER AND THE SOFTWARE (HARDWARE) NETWORK ARCHITECTURES TO BE USED?

10. DISTRIBUTED PROCESSING

    HOW WILL THE LANGUAGE SUPPORT DISTRIBUTED PROCESSING?

11. TRANSPORTABILITY

    HOW WILL THE LANGUAGE ADDRESS TRANSPORTABILITY CONCERNS?

12. LESSONS LEARNED

    HOW DO WE MAKE USE OF THE DATA ON LESSONS LEARNED ABOUT SOFTWARE MANAGEMENT?

INITIAL RECOMMENDATIONS FOR PANEL CONSIDERATION

1. REVISIT "HIGH ORDER LANGUAGE" WHITE PAPER (AUDREY DOROFEE)
2. USE ANSI STANDARDS
3. COLLECT DATA ABOUT DEVELOPMENT TO DETERMINE EVOLUTIONARY APPLICATIONS
4. ESTABLISH GENERIC REQUIREMENTS OF TOOLS
5. STANDARDIZE ON LANGUAGE - STUDY ADA
6. USE OF ASSEMBLY LANGUAGE SHOULD BE MINIMIZED
7. EVALUATE ADVANTAGES AND DISADVANTAGES OF A CANDIDATE SET OF LANGUAGES
8. EVALUATE DISTRIBUTED PROCESSING MACHINES WITH RESPECT TO LANGUAGES AND TOOLS
9. EVALUATE LANGUAGES FOR REQUIREMENTS AND SPECIFICATION, DESIGN, AND SPECIAL APPLICATIONS

Basili proposed that the panel proceed by (1) generating a set of goals based upon the requirements for Space Station, (2) refining (and defining) those goals for the various languages into a set of technology questions that should be answered, and (3) selecting languages or giving selection criteria based upon the answers to these questions.

Sample goal areas include theoretical, technical, methodological, political, management, and application oriented issues. Sample questions in these areas (adapted from questions posed by Susan Gerhart on Prolog) are:

Theory:

- Is the language well defined?
- What are the functional capabilities of the language and its limitations?

Technology:

- How stable is the technology behind the language design, the compiler design?
- Are there production quality compilers or interpreters?
- Are there performance issues that need to be addressed?
- Are there adequate development environments?
- How does the technology behind the language compare with the technology behind other languages in its class?
- What kinds of tools exist?
- Is there control of the definition of the language?

Methodology:

- What methodologies does the language support?
- Can the language be combined or interfaced with other languages and systems?
- Will the programs in the language make use of existing software in other languages?
- How are the usual desirable properties of programs, such as correctness, robustness, efficiency, modifiability, etc., addressed in the language?
- Can the language be integrated with other phase languages across the entire life cycle?

. How are other technologies supported by the language, e.g. transportability, distributed processing, prototyping, etc.

Applications:

. What application areas does the language address?

. What application libraries exist?

. What application areas have used the language?

Management:

. How does one manage (plan, control, direct) projects in the language?

. Can modern software engineering practices be brought to bear on projects in the language?

. What is involved in the training of personnel in the language?

Evaluation:

. Are there marketing and technical projections for the language?

. How does one become proficient in the language?

. What evaluations or case studies have been done, and what are the concerns and benefits they point out?

Social, Political, Historical:

. Is the language politically sound?

. What controversies has it gone through?

. What is the extent of its use?

RATIONALE FOR RECOMMENDATIONS

1. Recommendation:

NASA should avoid premature commitment to hardware implementation decisions. System and software architecture should be defined first.

Rationale:

A recurring problem with large systems, particularly those with a long development cycle, is that the hardware is selected (or mandated) before the system architecture is designed. As a result, the software architecture is overconstrained, memory and performance become serious constraints as the requirements evolve, and the hardware is obsolete before the system is operational.

By delaying selection of the hardware until the system and software architecture is understood, NASA can make intelligent engineering trade-offs between hardware and software. System and software architecture should allow

- early prototyping using available hardware or emulation,
- use of the most advanced hardware available when it is time to commit, and
- replacement of this hardware later with minimum impact.

This recommendation complements the SDE panel recommendation that the SDE support multiple targets. It does not conflict with the aggressive adoption of standards; rather, it serves to focus on adoption of standards at the appropriate level (e.g., bus standards and protocols). It is also consistent with the choice of Ada as the implementation language, provided that portability guidelines are developed and stressed.

2. Recommendation:

NASA should declare Ada now as the preferred high-order language for source code development and address the following issues as quickly as possible:

- transition strategy
- procurement issues
- interfaces to existing NASA software
- development of guidelines for applying Ada to various application areas
- development of appropriate run-time support environments for NASA applications
- education
- a liaison to DoD

- a seat on the Ada board
- benchmarks for performance
- prototyping
- development of appropriate tools to partition and allocate Ada entities across distributed applications
- introduction and utilization of reusable components
- investigation of fallback position options for high risk areas

Rationale:

Many aspects of Space Station software would be simpler if it were all written in a single programming language: compilers, support tools, training, software reusability, maintenance. Such uniformity is of course not completely realizable, for no single language would be appropriate in every case, and NASA already has software in several languages. Nevertheless, selecting one high-order language as the preferred language for new software and supporting this choice with the SDE and training would focus the Space Station software effort and foster the aforementioned benefits of commonality. Calling this selection a preference instead of a requirement would leave room for NASA to allow the use of other languages when it is more appropriate, while firmly establishing the direction of NASA's economic and organizational support.

If a single high-order language is to be preferred, it should be evaluated according to several criteria outlined elsewhere. One of these criteria is support for modern software engineering methods. It would certainly be a mistake for NASA to prefer a language that did not support these methods, for such a language would inevitably tend to impede their use. A high-order language supporting abstraction, information hiding, communicating sequential processes, and similar concepts would be a welcome improvement over older languages that do not adequately support these methods.

After reviewing the alternatives, the panel concluded that Ada is the language showing the greatest potential in this regard. Ada's strong data typing, packages, generics, and overloading support abstraction and information hiding. The exception handling capability supports the reporting and handling of errors and unlikely situations in a manner consistent with abstraction. Tasking supports communicating sequential processes at a higher level (analogous to procedure call) than other synchronization mechanisms, such as semaphores. Arithmetic is well defined and supports efficient fixed-point operations. Representation clauses support interrupt handling, hardware input-output interfaces, and similar implementation-dependent matters. Separate compilation supports efficient software development and distribution. Although Ada is a large and complex language, its features are useful.

Unlike most languages supporting modern software engineering methods, Ada is not a product of the academic community, with informal support and uncontrolled changes; nor is it a proprietary language with limited availability. Ada is a government and ANSI standard, and as such it is stable and supported. This support is rapidly growing. More and more compilers and programming environments for various host and target machines are coming onto the market. Applications are also being written in Ada. (The company of one panel member has already generated more than one million lines of Ada code.) Resources supporting Ada application developments are already in the range of one half billion dollars per year. The research community has taken a great interest in Ada and distributed systems, program design, program validation, and other areas applicable to Space Station. By selecting Ada, NASA can capitalize on this substantial investment and begin to influence the course of future Ada work.

If NASA is to choose Ada, it should do so now, so that activities dependent on this choice can begin. NASA and contractors need time for education, planning, and the specification of Ada-related requirements. Ada vendors need time to become aware of the new market provided by Space Station and to adapt compilers and run-time support packages to Space Station requirements. NASA must also address the series of issues enumerated in this recommendation, which are discussed below.

The first thing NASA must do is to formulate a strategy for the transition to Ada. Naturally, these plans will involve the management, standards, and SDE issues considered by the other panels. In particular, the SDE must include a full set of software development tools compatible with Ada.

Procurement issues must be addressed, including

- . development of Ada compilers and run-time packages for new environments
- . contractual obligation to use Ada and the SDE -- who will bear the risks?
- . contractual obligation to use Ada properly -- how can the use of appropriate software engineering methods be guaranteed?
- . waivers -- when is another language preferable for new software?
- . procurement of off-the-shelf software -- should it too be written in Ada, in case NASA should have to take over its maintenance? How would this affect its cost and availability?

NASA must decide how to apply its large base of existing software to Space Station:

- NASA could continue to use stand-alone software, as long as maintenance costs were not excessive.
- Other software could be used directly within an Ada environment, if suitable implementations of the "interface" pragma existed in that environment. NASA would probably have to fund the development of Ada interfaces to HAL/S and any other NASA-specific languages. Perhaps it would be better to rewrite such software in Ada: this would often be straightforward, the resulting Ada code would be much more portable, and it might even be economical if the software had to be changed anyway.
- Software that is not directly reusable may contain the only existing documentation for algorithms applicable to Space Station. Important algorithms that would be difficult to re-derive should not be lost; Ada or an Ada-based PDL would be an ideal medium for preserving and documenting them, as well as using them.

A caveat is in order, however: Most older software was developed without benefit of concepts that enhance reusability and ease of change, such as abstraction, information hiding, and even good documentation. Consequently, the strategies noted above may prove difficult. Old software should be evaluated and adapted using the same criteria applied to new software; to do otherwise would defeat much of the purpose of using Ada and would prolong reusability, portability, and maintainability problems into the 21st century.

Training in the proper use of Ada is of such importance that the panel made a separate recommendation in this area (see recommendation 3).

Any major Ada user should have close ties with the Ada community at large. Consequently, NASA should establish a liaison with DoD and the Ada Joint Program Office. Furthermore, any agency committing such an important and visible project to Ada deserves a voice in Ada's future development. Therefore, NASA should seek a seat on the Ada board.

Currently, Ada compilers are validated by the DoD with respect to correctness only; they do not have to pass any performance benchmarks. Since performance will be a major issue in many Space Station applications, NASA should initiate or jointly sponsor a benchmarking activity for evaluation of Ada compilers and support libraries. It should test the performance of Ada programs in distributed systems and high-speed real-time systems as well as in more routine contexts. Such benchmarks will also help to identify high-risk areas needing attention.

Ada's support for abstraction and information hiding makes it especially good for rapid prototyping. Once a design has been produced in the form of a collection of Ada package specifications (with associated semantics), the component packages can be implemented in parallel, each without regard for how the others are implemented. Such a prototype can then be transformed into a finished product by independently changing the implementations of each of its components. With the interface pragma or a special interface package, the SDE might also support the rapid implementation of an Ada package using a separate program, perhaps in a very high level language (e.g., Prolog). NASA should use early prototyping to investigate application areas such as fault-tolerant and distributed systems. This would help determine how well Ada supports these applications and would consequently reduce the present uncertainty in this regard.

The use of Ada in distributed systems, including the need for tools to allocate Ada entities across such systems, is addressed further in recommendations 4 and 5.

Space Station applications should share the same software wherever possible. Reusable software can reduce the cost of software requirements specification, decomposition, and design (because it is often easier to recognize what is needed than to define it), coding and testing (because neither is needed in order to use an existing, tested implementation), and maintenance (because changes to one reusable module are cheaper than changes to several nearly identical ones). Ada is an excellent tool for supporting reusability, since reusability is directly related to abstraction and information hiding. However, it is no trivial matter to design abstractions that are amenable to reuse. To support reusable software, NASA should

- **develop or adopt a taxonomy of software abstractions,**

- **identify specific reusable abstractions,**

- develop a library of Ada package specifications for these abstractions, catalogued according to the aforementioned taxonomy (so that projects can find packages useful to them)
- develop a prototype package body for each library package (so that projects can test code that uses these packages),
- publicize the library and encourage — perhaps even reward — the use of its packages,
- develop efficient package bodies for each library package (so that projects can test their products for performance and release them), and
- devise a plan for adding to this software library.

The SDE should support the use of reusable components from this library and the search of the library catalog for components of interest. In addition, it should allow the library to contain more than one implementation version of a single Ada package, so that users can select from implementations optimized in different ways (e.g., execution speed versus memory required).

The choice of Ada is not without risk, although much of it is in areas that will be risky whether Ada is used or not. In particular, in some quarters there is uncertainty about (1) the applicability of Ada to distributed, fault-tolerant, and hard real-time systems, (2) the efficiency of Ada run-time support environments and of code generated by Ada compilers, especially for tasking in real-time and distributed systems, and (3) the development of good Ada implementations for the particular machine architectures that might be used for Space Station.

Prototyping, benchmarking, and work on run-time support environments should resolve the first two issues. Postponement of hardware selection and the eventual use of off-the-shelf machine architectures should minimize the last problem, by reducing the chance that an unexpectedly difficult architecture will be selected with insufficient time to produce a good implementation for it. Nevertheless, at least until these problems have been put to rest, fallback policies should be established in each of these problem areas.

3. Recommendation:

The commitment to Ada requires an education program in software engineering methodologies with Ada, which should begin as soon as possible. The education includes the study of relevant examples. It should cover multiple levels of management, application programmers, etc.

Rationale:

The rationale behind this recommendation may be perceived from three perspectives: systems engineering, methodology, and language.

From a systems engineering perspective, Space Station software is just one important part of a complex system. Software management, development, acquisition, and evolution are all subordinate to a total systems engineering activity requiring management and technology trade-offs. These trade-offs are constrained by practices, obligations, and requirements at the project, systems, subsystems, and institutional levels. To make intelligent decisions at all levels of management and engineering, NASA personnel need to understand, to different degrees and from different perspectives, the programming and engineering capabilities and limitations of Ada, and the management implications of using Ada.

For example, if it went uncorrected, the myth that Ada is inherently inefficient could distort evaluations of trade-offs between hardware and software, or between Ada and some other programming language. As a more positive example, an understanding of how Ada and the technique of information hiding can support abstract interfaces to hardware would make the strategy of postponing hardware selection appear much more practical.

From the methodological perspective, Ada is more than a mere programming language. It embodies and supports modern software engineering concepts, such as rich data structures, data abstraction, information hiding, modular packaging, exception handling, and communicating sequential processes. It has features that enforce disciplined engineering, such as strong typing. It (or a derivative) can be used as a high-level program and system design language. It is to be used in conjunction with an Ada Programming Support Environment comprising powerful tools for software development. Together, these form a system supporting modern software engineering methods. To ensure that developers and contractors take full advantage of these methods and Ada's support for them, NASA personnel must themselves understand them.

Although the aforementioned software engineering concepts are well known in the academic and research communities, they have not penetrated the software community at large to any great degree. Consequently, many software professionals will come to the Space Station project without experience in applying these concepts, and sometimes without even a basic understanding of them. Therefore, NASA will need a training program that provides

- good definitions of these concepts,
- examples of their use, and
- practice in applying them to program design and implementation with Ada.

The shortage of professionals trained in these methods extends to the education and training community itself, so NASA should establish a quality assurance program to guide and audit this training.

For example, the important concepts of abstraction, information hiding, and communicating sequential processes can be briefly defined and related to Ada as follows:

- Abstraction supports the orderly decomposition of a software system into components that can be understood solely by reference to their interface specifications, which include black-box descriptions of the associated behavior; implementation details are suppressed. In addition to facilitating the program design process, this enhances software reusability, since each abstraction is a potentially reusable design. Each of Ada's compilation units (package, task, subprogram, and generic) supports a kind of abstraction.

- Information hiding emphasizes the importance of concealing the details of the implementation of an abstraction. Because these details are hidden, users of the abstraction cannot make unwarranted assumptions about the implementation; this makes it easier to change the implementation without affecting the software that uses it. Information hiding involves designing the abstractions used to build a system so that each aspect of the system that is judged likely to change is hidden behind a single abstraction; by anticipating changes, it makes those changes easier. Ada's packages, visibility rules, and private types support information hiding.

- Communicating sequential processes (CSP's) allow the decomposition of a system into tasks that logically run in parallel, occasionally communicating with one another. Complex real-time systems can be built using CSP's, and distributed systems can be implemented by assigning CSP's to different processors; however, many more mundane problems also have natural solutions involving CSP's. Ada's tasks support communicating sequential processes.

Viewed simply as a rich language, Ada can either be applied properly to solve complex problems, or it can be misused to complicate solutions. A programmer experienced with conventional languages may be tempted to use Ada as a conventional language with new syntax. This mode of application would be most unfortunate, for it would defeat the fundamental purpose of Ada's existence, which is to foster the use of methods mentioned above. To fully exploit Ada's many features, programmers (both NASA personnel and contractors) need training on its proper usage.

The study of relevant examples will be an important part of all this training. Obviously, examples of Ada programs will be relevant in this case. However, bad Ada programs should not be used as examples — other than examples of what not to do. Unfortunately, there is a real danger here: some books on Ada utterly fail to address the software engineering principles that Ada was developed to support, and instead teach little more than mechanical translation of bad programs in other languages into bad programs in Ada.

On the other hand, some of the best and most relevant examples may not even use Ada. Examples of good software engineering methods are rare, and fully worked out examples of systems of reasonable size are rarer still. Some of these may use other languages, but they will nevertheless be worthy of study by those involved in software design, for it is the method of decomposing software into modules and defining the interfaces of those modules — the software architecture of the system — that is the most important aspect of an example. A good architecture will be valid regardless of the implementation language, and it will be easy to map into Ada.

NASA should search the literature for examples of good software design applicable to the use of Ada before trying to develop them in house or under contract. Even if an example is not fully implemented, it may still contain useful material.

At this time the pool of trained Ada professionals, particularly lead designers, is very small. The typical training time for a lead designer may be as much as a year. NASA must rapidly select or develop training methods that will ensure a sufficient supply of trained professionals for the Space Station program. Training may prove to be the largest startup cost of the transition to Ada. The duration and success of this training will have a strong effect on the long software life cycle projected for Space Station.

4. Recommendation:

NASA must define its requirements for the run-time support library and kernel for the target systems, including distributed targets.

Rationale:

To derive the maximum benefits from the choice of Ada as the preferred high order language for source code development, NASA should move quickly to determine and catalog its requirements for the run-time support environment of target processors to be embedded within the applications needed for the Space Station program (e.g., highly data-driven applications versus critical, real-time applications). Although such requirements are not unique to NASA, the panel feels that

. the development schedule for the Space Station program plus

. the lack of an appropriate catalog of requirements for the run-time support environment of processors embedded in large, complex, distributed applications

should cause NASA to quickly take a leading role in defining such requirements.

The run-time support environment (RTSE) provides resource management and other services to the object code modules of the application programs. This support is typically provided by a run-time kernel, which separates both the application modules and the run-time library modules from the bare target processor. The kernel contains a minimal set of functions that are used frequently and must be executed sequentially.

The run-time library may be divided into a basic library set and an extended library set. The basic library contains modules that provide services to the object code modules produced by a host Ada Programming Support Environment (APSE) for a broad class of applications. The full set of basic library modules need not be present on all target processors. For example, if the application program objects assigned to a given target processor do not require Ada's tasking or heap management, then the basic library modules responsible for those facilities may be omitted from the run-time environment.

The extended library contains modules that may be used to support APSE-produced object code in specific applications having requirements beyond those addressed in the Ada Language Reference Manual (ref. 3). For example, many applications would benefit from a run-time "monitor" that gathers and reports performance statistics and facilitates remote diagnostics and reconfiguration. Other modules might support multi-level security and access control, or transactions with nested atomic actions. All such modules could be transparent at the Ada source code level and thus facilitate the cost effective utilization of reusable components across a broader spectrum of applications.

Clearly, regardless of the efficiency and reliability of the object code produced by a host APSE, the performance and reliability of the executing program are dependent on the run-time kernel and library.

Another important reason why NASA should begin quickly to define its requirements is the complexity spectrum of implementing RTSE's shown below:

- Single "stand-alone" embedded processor to support
  - subsets of Ada
  - full Ada
- Multiprocessor applications, which support the partitioning and allocation of objects within the Ada programs for execution in target environments implemented with
  - shared memory
  - shared bus
  - "n level" redundancy
  - combinations of the preceding
- Distributed network applications, which support the partitioning and allocation of objects within the Ada programs among geographically separate processing resources for execution. Such implementations may include
  - Local area networks composed of single processor nodes and multiprocessor clusters
  - Remote area networks of local area networks, single processor nodes and multiprocessor clusters.

Ada was designed to serve as a "common language for programming large scale and real time systems" (Foreword, ref. 3). The objects of an Ada program can be distributed "whenever an implementation can detect that the same effect can be guaranteed" as for execution by a single processor (Section 9, ref. 3). However, the current implementations of Ada compilers and environments respond only to the requirements for a Minimal tool set (MAPSE). Those requirements address a single, stand-alone target processor, and therefore only the simplest RTSE on the complexity scale. Specifically, the MAPSE does not require the tools needed for

- allowing the software engineer to scan the Ada source code and identify which program objects should be allocated to which target resources and then
- building the load modules of application code and, possibly, run-time library modules to be exported to the various target processors.

(It should also be noted that such tools have not been created in the HAL/S environment or in other environments that were not designed to support large, complex distributed applications.) The construction of such tools as a necessary part of the Space Station program's Software Development Environment is dependent upon an understanding of NASA's requirements for a catalog of features and options for the run-time kernel and run-time library.

5. Recommendation:

NASA needs to define the requirements for the interface to the run-time system.

Rationale:

Whereas recommendation 4 addressed the need for NASA to begin defining its spectrum of requirements for the functionality, performance, and reliability of the run-time support environments needed for the Space Station program, this recommendation focuses specifically on the requirements for the interface of the object code of the application programs to the run-time kernel and run-time library.

A major goal of the Space Station program is to support technology transparency. The economics of thirty or more years of Space Station evolution, operation, and maintenance will require that diverse instruction set architectures (ISA's) coexist in the target environment. Some of these ISA's will participate in subsystem activities that provide an integrated, end-to-end information system from earth stations, through entities in various earth orbits, to a permanent presence on the moon. Some of them will participate in subsystems that must operate continuously during diagnostics, repair, expansion, reconfiguration, software and hardware updates, and other system activities. Thus, the ability to map the object code modules of applications programs to an interface model of a virtual Ada machine is highly desirable.

Hiding machine dependencies as much as possible (consistent with NASA's requirements for RTSE functionality, performance, and reliability) and encapsulating code that must be machine dependent will enhance the transportability, reusability, and interoperability of Ada source code modules and thus help control the costs of software ownership and incremental development.

Organized, international working groups are now addressing these interface issues. NASA should take a leading role in advancing this work.

6. Recommendation:

The first version of the SDE should not be constrained to have a single requirements language, AI expert systems language, or prototyping language.

Rationale:

There are a number of requirements methodologies, languages, and tools that might be of use for Space Station software development. The panel considered whether NASA should select a preferred or standard set of requirements languages, to facilitate communication among space station participants and contractors. However, the panel decided not to recommend this because

- Space Station needs in this regard are not yet well defined;
- requirements methodologies, languages, and tools have not yet reached the degree of maturity required for selecting standards; and
- it is not clear that any of the currently available items is adequate for Space Station needs.

Similar considerations make it premature to select other specialized languages, such as expert system languages and prototyping languages.

However, because all these types of development aids have potential for improving the productivity of the software and system life cycle, their use should be explored. For this purpose, the SDE should initially offer a selection of languages of each type. Many of the criteria for language selection given elsewhere should be applied to the evaluation of these languages. [See also recommendation 7.]

7. Recommendation:

NASA should determine the requirements for, and select or develop requirements and design specification languages or interfaces that complement the SDE and Ada.

Rationale:

The specification of both system and software requirements and designs for a system as complex as the Space Station is a major undertaking that is crucial to system success or failure. Previous programs at NASA and elsewhere have identified requirements specification in particular as an extremely difficult activity in the system life cycle. It often has been characterized as a chaotic decision-making process exacerbated by a lack of adequate methods, languages, notations, and tools. Research and development efforts over a decade or more have resulted in a number of approaches and tools, some of which have merit for the Space Station effort.

The panel considered whether NASA should simply rely on existing languages and tools to meet Space Station needs. The panel did recommend that several of these aids should be part of the initial SDE [see recommendation 6].

However, the magnitude of the Space Station undertaking and the benefits of good requirements and design specification aids argue strongly for a focused, early effort to define and then acquire a set of tools tailored to meet the specific needs of the Space Station program. The panel expects that many of these tools will be commercially available, but some may have to be developed. NASA's goal should be a set of standardizable requirements languages and interfaces that can be used to facilitate communication among all Space Station participants.

If Ada is to be the primary software implementation language, then any requirements and design methods eventually adopted should be consistent with the use of Ada. (Recommendation 8 addresses program design languages in more detail.) Similarly, SDE support for these methods is crucial if they are to be used efficiently and in a disciplined manner.

8. Recommendation:

The design language should be syntactically and semantically consistent with the development language and should have on-line support for interface checks, etc.

Rationale:

The use of a program design language (PDL) is a recognized component of good software engineering practice. A common excuse for avoiding the practice is that, as the software evolves, the PDL is an added cost and often becomes inconsistent with the code.

These difficulties can be overcome if the PDL is consistent with the development language because the PDL is embedded in the implementation. As such a software structure evolves, the PDL is maintained naturally. Further, designs using such a PDL can be checked for semantic consistency.

Given Ada's facilities for structuring software, the use of an Ada-compatible PDL would allow semantic consistency to be maintained throughout the software implementation. The SDE should therefore support the use of an Ada-based PDL. The IEEE is currently completing a standard for the use of Ada as a PDL; NASA should investigate whether this standard is appropriate.

In cases where Ada is not used as an implementation language, an explicit decision should be made whether to use Ada as a PDL or to use a PDL consistent with the implementation language. In fact, this decision should be considered in the trade-off analysis leading to selection of a language other than Ada for a particular application.

9. Recommendation:

For all levels of user interfaces, there should be a set of standards to provide commonality across all phases of the Space Station life cycle.

Rationale:

The need for a set of standards for user interfaces is driven by the following considerations:

- the long life cycle of the Space Station and its support systems and environments,
- the constantly changing and growing set of users,
- the use of common or government furnished support systems and environments,
- the need to minimize program costs, including software, training, and customer costs,
- the high degree of commonality in the functions performed by various types and groups of users, and
- the high degree of coordination and integration of activities and products required throughout the program.

A set of standards for user interfaces (i.e., methods and languages) will

- permit users to migrate among sites and across support systems and environments without the need for extensive retraining,
- provide a greater degree of portability and reusability of user generated procedures and programs,

- decrease communications, coordination, and data exchange problems among user groups,
- provide a central core to which unique user interface requirements can be added, and
- minimize the amount and cost of user interface software, documents, tools, and training.

10. Recommendation:

NASA should identify all categories of users and user interfaces, and quickly proceed with rapid prototyping to determine the real requirements.

Rationale:

User interfaces are an essential part of any support system, environment, or tool. The definition and design of user interfaces come early in the life cycle of support systems, environments, and tools. If there is to be a set of standards for Space Station user interfaces (as in recommendation 9), all user categories must be identified, and their interface requirements must be defined and analyzed to derive that set of standards.

To be of maximum benefit to the program, these standards must be ready in time to be applied to the work that has already begun on common and government furnished support systems and environments. These systems and environments will not only have their own user interfaces, they will also support the development of software, tools, and systems having still more user interfaces. It is therefore imperative that users and user interface requirements be identified as soon as possible.

Rapid prototyping would probably be the most viable method leading to the definition of interface requirements and the derivation of standards.

11. Recommendation:

Since Space Station software will evolve over 30 years, NASA should track language technology and act appropriately.

Rationale:

Thirty years is an unprecedented lifetime for software. No project of such duration should ignore the advance of relevant technology. Developments in software technology over the past thirty years -- especially those of the past decade -- presage even greater changes during the next thirty.

Some phases of the software life cycle do not have good language support at this time. The requirements definition phase is a case in point; should better language support emerge for requirements definition, NASA and the Space Station project would surely benefit from it. Similar reasoning applies to aspects of software outside the traditional life cycle, such as prototyping.

On the other hand, obstraction and information hiding will in any event continue to be fundamental principles for structuring software. This generality is important, because it supports the decomposition of a software engineering problem into subproblems that can be implemented independently, each in the most appropriate language. For instance, it should eventually be a straightforward matter to implement an Ada package specification as a program in a fifth-generation artificial intelligence language. This sort of flexibility should also be a goal of this SDE.

Languages evolve to support software technology and consequently serve as indicators of the state of that technology. NASA needs to track all software technology; tracking language technology is an important subset of such activity.

N86-23318

SOFTWARE STANDARDS PANEL SUMMARY

The unique and challenging nature of the Space Station Program requires that software standards be effectively used to control costs, facilitate enhancements and ensure safety. The Software Standards Panel identified and developed recommendations in four areas to help the Space Station Program achieve these objectives. The areas in which recommendations are offered are policy, organization, process and candidate software standards for the Space Station Program. The consensus process employed by the panel involved:

A. Initial survey of general software standards issues.

B. Analysis of the specific software standards issues stated in reference 1.

C. Restatement of issues and discussion in open panel session.

D. Consideration of alternate recommendations.

E. Development, presentation and discussion of specific recommendations in open panel session.

A list of the recommendations arrived at in the above manner is given in the following section. The panel did not attempt to recommend the selection of specific software standards, but did recommend that NASA move at once to act on the selection of standards in specific areas. A minority of the standards panel, as well as large number of audience participants, took the position that current software standards have grown into areas that are not consistent with the traditional concept of standards. In other words, the current definition (usage and implementation) has been bent far beyond a useful definition of "standards". A critical re-examination of standards, at this time, would be in order.

RECOMMENDATIONS

The Software Standards Panel recommends that the Space Station Program Office take the following actions:

1. Establish a Program policy supporting software standards.

2. Establish an organizational structure to support software standards at each level within the Space Station Program.

3. Capitalize on existing software standards to meet Program requirements.

4. Establish software standards early in specific candidate areas.

D4

N86-23318

NASA Space Station Software Standards Issues

George D. Tice, Jr.
Tektronix, Inc.
P.O. Box 4600 (M/S 92-525)
Beaverton, Oregon 97075
(503) 629-1310

ABSTRACT

The selection and application of software standards present the NASA Space Station Program with the opportunity to serve as a pacesetter for the United States software in the area of software standards. This presentation summerizes and discusses the strengths and weaknesses of each of the NASA defined software standards issues:

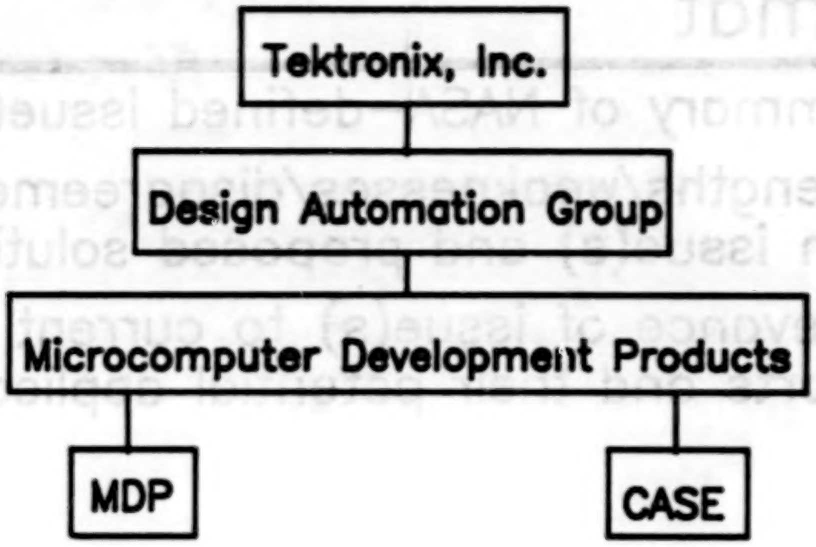
- Need for Common Software Terminology
- Project Directives
- Software Technology
- Software Portability
- Languages
- Documentation

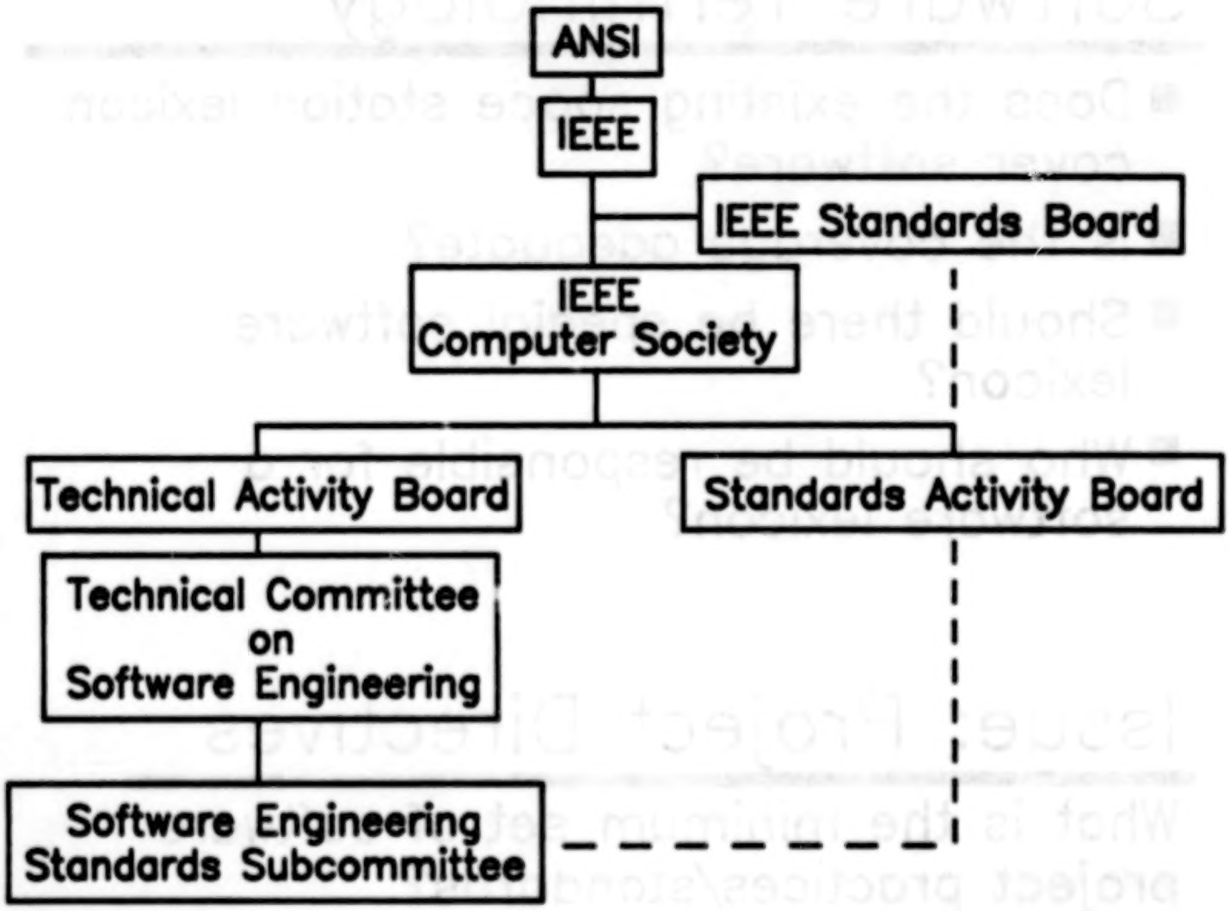
Several additional significant standards issues are offered for NASA consideration:

- Value of Standards
- Potential Leverage from Other Standard Efforts

The presentation concludes with a challenge for the NASA Space Station Program to serve as a pacesetter for the U.S. Software Industry through:

- Management commitment to software standards
- Overall program participation in software standards
- Employment of the best available technology to support software standards


Tektronix, Inc.
Design Automation Group
Microcomputer Development Products
MDP
CASE


ANSI
IEEE
IEEE Standards Board
IEEE Computer Society
Technical Activity Board
Standards Activity Board
Technical Committee on Software Engineering
Software Engineering Standards Subcommittee

Format

- Summary of NASA–defined issue(s)
- Strengths/weaknesses/disagreements with issue(s) and proposed solution(s)
- Relevance of issue(s) to current R&D efforts and their potential application

Issue: Need for Common Software Terminology

- Does the existing space station lexicon cover software?
- Is the coverage adequate?
- Should there be special software lexicon?
- Who should be responsible for a software lexicon?

Issue: Project Directives

What is the minimum set of software project practices/standards?

Issue: Software Technology

- What criteria should be used to select SSIS software technology?
  - software engineering and practices
  - standards for portability
  - programming language
  - whether to impose an instruction set architecture
  - data driven vs. data embedded software
- What criteria should be used for technology changeover?
- How can we make technology change transparent?
- How do we keep current in technology?

Issue: Software Portability

Applicability and methodology of portability and transferability for the space station.

Issue: Languages

Languages for software development?

Issue: Documentation

- What is the critical, minimal set of documentation and what level of detail should be specified?
- Do the critical set of documents and level of detail vary with software category?
- What acceptance criteria are needed?

Additional Significant Standards Issues for NASA Consideration

Issue: Value of Standards

- Education
- Simplification
- Conservation
- Certification
- Contribution

Issue: Potential Leverage from Other Standard Efforts

- Department of Defense (DoD)
- European Space Agency (ESA)
- IEEE Software Engineering Standards

NASA Space Station Program

Challenge and Opportunity

Serve as a Pacesetter for the U.S. Software Industry

- Management commitment to software standards
- Overall program participation in software standards
- Employment of the best available technology to support software standards

DISCUSSION OF RECOMMENDATIONS

1. NASA Space Station management should establish policy supporting software standards which:

   A. States top level (levels A & B) endorsement and commitment.

   B. Defines implementation and enforcement authority and mechanism.

   C. Provides methodology for software standards training and encourages its use.

   D. Provides an overview (audit) program to measure effectivity and encourage adherence to software standards.

   E. Encourage technology infusion/insertion.

To be effective, standards must have top management's unconditional support and that support must be visible at all levels of activity. Unless the purpose of each standard is understood and the methodology for selecting, implementing and enforcing standards is known to be rational, they will be viewed with suspicion. It is necessary to continuously maintain the currency of software standards to ensure their utility, and thereby their continued use.

2. The NASA Space Station Program should establish a structure to develop and support software standards having the following characteristics:

   A. Level A management authorizes the structure to support software standards.

   B. A Space Station Software Standards Organization at level B with responsibility for promulgating, maintaining and enforcing software standards.

   C. A Software Standards Advisory Committee with level C representation to advise the Software Standards Organization on the need, feasibility and acceptance of proposed changes to software standards.

The mechanism for supporting software standards must be structured such that issues can be resolved at the appropriate levels. It must remain in constant touch with the user community to understand their requirements for and problems with software standards. It must be flexible enough to act quickly when change is needed and strong enough to resist change when that change will weaken the overall system of standards.

3. The NASA Space Station Program should proceed to acquire standards as follows:

   A. Establish a need for software standards based on Space Station system/software requirements.

   B. Establish a standard for standards to promote understandability and improve communication.

   C. Establish a priority for source selection of standards (international, ESA, industry, NASA, contractor, etc) that supports both the objectives and needs of the Program and organizations involved in it.

D. Review existing software standards in light of requirements and priorities.

E. Select and tailor from existing standards when possible and develop new standards as a last resort.

F. Implement new standards on a trial basis with specific criteria for rejection and full implementation.

As with any system, it is critical that the most essential elements of the Space Station software standards system be identified early so that they may be brought into being in the proper sequence. This will enable the needed standards to be available at the appropriate time and avoid a bottom-up muddle of incompatibility.

4. The NASA Space Station Program should immediately act to satisfy its needs for software standards in the following areas:

A. Common Software Terminology (Lexicon)

B. Software Engineering Methodology and Practices

- o Software Management
- o Software Acquisition
- o Software Development
- o Coding
- o Documentation
- o Measurement and Data Collection

C. Languages

D. Instruction Set Architecture

E. Networks

F. Operating Systems

G. Applications (e.g. DBMS)

H. Security

These candidate areas represent the fundamental variables that must be managed and controlled through standardization to provide for a cost-effective software acquisition and support activity.

CONCLUSIONS

Results of the panel deliberations were summarized at the close of the forum. Subsequently, the panel members carefully documented their findings and these contributions are included in this publication. The panels identified many issues and provided recommended actions for NASA to consider. These are now under study by members of the Space Station Program.

It is noteworthy that the recommendations from the panels are practical and workable, rather than academic long-range forecasts (e.g., go with Ada, identify a mandatory SDE, begin with Unix).

Although each panel operated independently, establishing its own format and leading its open forum discussions, there were several common themes that emerged, as noted by representatives of the Software Working Group who attended each of the panel open sessions. Some of the common topics mentioned by more than one panel were as follows.

- Many issues, especially in the management and standards areas, apply to systems rather than just to software. Software should be developed and managed as an integral part of a systems level strategy.
- Incremental development methodology should be practiced.
- Interfaces between software components and between hardware and software should be identified early and then managed.
- Technology evolution must be accommodated over the Space Station lifetime.
- Selection of computer hardware should not be a limiting factor on the software.
- NASA has much experience with large software projects and should use the lessons learned from the past in this development. Also, provision should be made to capture lessons learned during the development and operation of the Space Station for the benefit of future projects and the continued evolution and growth of the Space Station itself.
- Focus on maintenance, plan for it from the beginning.
- Begin training in software areas early (e.g., Ada programming, SDE use, software management procedures).

Some common concerns were also expressed during the various open sessions

- Schedule constraints indicate there is very little time for good "front-end" work.
- Many software terms are subject to individual interpretation and should be specifically defined, such as rapid prototyping, incremental development, users, languages and tools, risk management, life cycle, use of Ada, training. (Note that the proposed Space Station Software Lexicon will address this concern.)
- A potential incompatibility exists between "design-to-cost" and "life cycle costing", since it may not be possible to simultaneously optimize front-end and back-end costs.

PAGE 126 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

SOFTWARE MANAGEMENT PANEL CONCLUSIONS

The Software Management Panel agrees with NASA's assessment of the critical importance of software to the success of the Space Station Program. This is exemplified by the implementation of NASA-wide software cognizant organizations and support functions. The early production and substantial content of the draft Level A/B Software Management Plan are indicative of the ability and viability of the software organization. These are good beginnings.

The panel members have reviewed and deliberated on the Space Station Software Issues Report and the Software Management Plan. The resultant views were merged with the perceptions of industry and NASA invited representatives in open forum to produce a substantial and well-founded set of recommendations. These recommendations will facilitate further progress toward a meaningful, operative Software Management Plan. Given NASA's commitment to software excellence and the unique technical challenges of Space Station, the following conclusions are clear to the panel.

1. The necessary tasks and responsibilities envisioned for the Level A and B software management organizations far exceed their current and projected resources and authority. This is particularly the case at Level A. Resolution of this situation is fundamental to the success of the Space Station Program.

2. NASA needs to expand its excellence from in-house engineering to the arms-length acquisition of software in many categories to operate together in a very large system.

3. The focus of software management and acquisition should shift to maintenance/sustaining engineering in order to minimize life cycle costs.

4. Management procedures for interaction with non-Space Station services and users should be properly defined.

5. The scope of the top level software engineering and integration of Space Station software should be assessed and addressed.

6. The Software Management Plan should be restructured to recognize the foregoing needs.

SOFTWARE DEVELOPMENT ENVIRONMENT PANEL CONCLUSIONS

The concept of a uniform SDE furnished and mandated by NASA, to address the critical life cycle cost and integration issues of Space Station software, is strongly endorsed. Risks, such as schedule, technological obsolescence, and contractor incompatibilities, can be mitigated by an incremental acquisition strategy, the use of layered architectures to assure technological transparency, and an operational concept which provides for contractor options to use their own SDEs, as long as the delivered software is supportable by the NASA SDE. This operational concept should be developed soon and should address user requirements and life cycle scenarios based on inputs from users, Phase B contractors, and similar DoD efforts (e.g., the JSSEE Operational Concept Document).

The SDE should focus on products (such as specifications, design/code representations, etc.) rather than specific methodologies, and should encourage the reuse of previously developed software in order to save costs.

The architecture of the SDE should be modularized and layered to allow for technological evolution at distinct levels. The operating system should be vendor and device independent insofar as possible. Unix appears to be the only candidate that meets these criteria, and should be considered as the initial basis for the operating system.

The SDE should be furnished as a portable software package (so that changes in hardware do not affect software functionality) and should consist of a subsetable set of tools engineered with uniform interfaces providing the capability to customize to specific user requirements by application (e.g., flight or ground software development, analysis, management, simulation), by type of user (e.g., expert/novice, specialist/generalist), or by type of equipment (e.g., mainframe, mini, or work station. A major objective is to maximize commonality of widely used functions. The SDE should automatically collect data that characterize its use, and these data can be used as the basis for improvement and extension of the SDE.

LANGUAGES PANEL CONCLUSIONS

The selection and standardization of languages and interfaces for the Space Station program are critical needs to insure the success of this predominately engineering activity. While the Language Panel recognizes that the project life cycle will require a family of languages for the various classes of users and developers, it is crucial to begin making decisions which will focus planning efforts by limiting the range of possible selections. Requirements for the Space Station information system long-term maintenance and evolution will mandate that a high-order development language be utilized. It is recommended that the primary high-order language for source code development be Ada. Issues related to the utilization of Ada should be addressed as soon as possible. These include developing a transition strategy, providing education, accommodating the utilization of software already in existence, and developing fall-back options for high risk areas. One high-risk area is satisfying the requirements for run-time support for target systems, especially when the targets are distributed. Requirements for design specification languages or interfaces that complement Ada should be determined.

Additional conclusions are that the premature commitment to hardware implementation decisions should be avoided, and that the critical need is to develop the system and software architectures for Space Station.

SOFTWARE STANDARDS PANEL CONCLUSIONS

Standards are one of several elements that provide a common "backbone" for all software aspects of the Space Station Program. The operational Station will eventually influence NASA's entire future space activities. Because of this broad area of impact, the importance of software standardization for the Space Station Program must not be underestimated.

This forum has resulted in recommendations that encompass the major aspects of the needed software standards program. The procedure for arriving at these recommendations ensured that NASA received the best possible advice available in the area of software standards. Due to the farsightedness of the Space Station Software Working

Group, there is adequate, but not excessive, time available to implement the Standards Panel's recommendations.

A unique situation and opportunity has been created. The Space Station Program has received needed advice from experts at a key point in the Program on a critical subject. That subject happens to be software standards, an area of technology that has been stalled for too long a period. NASA is in a position not only to establish an outstanding software standards program for the Space Station, but to provide the software industry with a much needed innovative model in this area. NASA should move at once to act on the recommendations provided.

Additional standards issues should be addressed including the distributed network operating system, graphics (Core vs. GKS), standards for device independence (VDI, VDM for storing graphics, IGES or NAPLPS for transmission), program/operating system standard interface, self-documenting data record format, and OSI communications protocol standards.

REFERENCES

1. Voigt, S. J. (Ed.): Space Station Software Issues. NASA CP-2361, 1985.

2. Military Standard Defense System Software Development. DOD-STD-2167, 1985.

3. Reference Manual for the Ada Programming Language. ANSI/MIL-STD-1815A-1983, 1983.

1. Report No. NASA CP-2394	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle SPACE STATION SOFTWARE RECOMMENDATIONS		5. Report Date December 1985
		6. Performing Organization Code 482-58-13-02
7. Author(s) Susan Voigt, Editor		8. Performing Organization Report No. L-16063
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665		10. Work Unit No.
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Adminstration Washington, DC 20546		13. Type of Report and Period Covered Conference Publication
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

The Open Forum on Space Station Software Issues was held at NASA Marshall Space Flight Center, Huntsville, Alabama, on April 24-25, 1985, with representatives from industry, government, universities, and a few foreign space agencies. Four panels of invited experts and NASA representatives focused on the following topics: software management, software development environment, languages, and software standards. Each panel deliberated in private, held two open sessions with audience participation, and developed recommendations for the NASA Space Station Program. The major thrusts of the recommendations were as follows. (1) The software management plan should establish policies, responsibilities, and decision points for software acquisition. (2) NASA should furnish a uniform modular software support environment and require its use for all space station software acquired (or developed). (3) The language Ada should be selected for space station software, and NASA should begin to address issues related to the effective use of Ada. (4) The space station software standards should be selected (based upon existing standards where possible), and an organization should be identified to promulgate and enforce them. These and related recommendations are described in detail in the conference proceedings. Several common themes also emerged. These were: learn from past experience, obtain "real" requirements for software support, identify and manage interfaces early, focus on maintenance as a primary requirement, include software as an integral part of the system level strategy, and define terminology.

17. Key Words (Suggested by Author(s)) Software management policy Software support environment Space station software Software standards Computer languages		18. Distribution Statement Unclassified - Unlimited Subject category 61	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 144	22. Price A07

For sale by the National Technical Information Service, Springfield, Virginia 22161 NASA-Langley, 1985